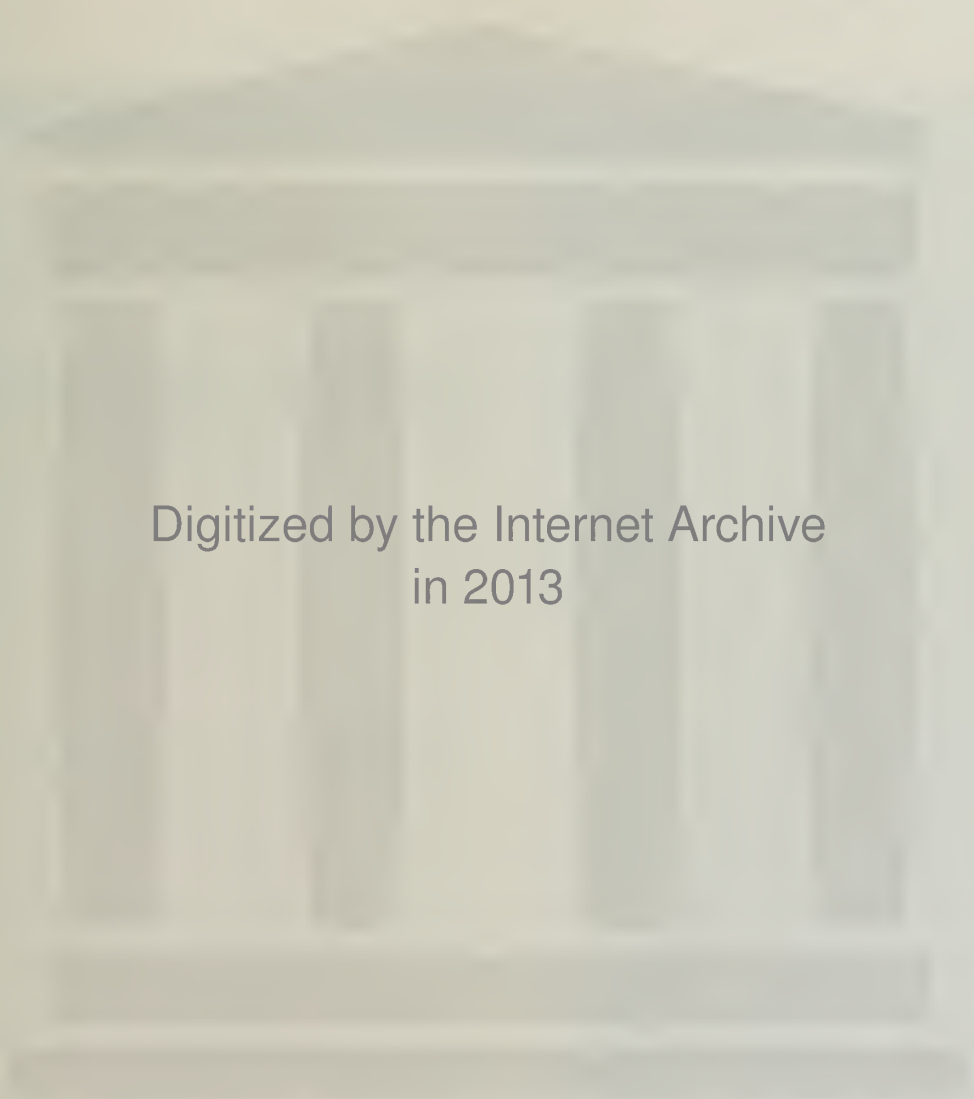


LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
I 26 r
no. 722-727
cop. 2





Digitized by the Internet Archive
in 2013

<http://archive.org/details/analysisofschedu724chen>

310.8-1
Illn
no. 724

Math

9

UIUCDCS-R-75-724

THE LIBRARY OF THE

JUN 13 1975

AN ANALYSIS OF SCHEDULING ALGORITHMS IN
MULTIPROCESSOR COMPUTING SYSTEMS

UNIVERSITY OF ILLINOIS
URBANA-CHAMPAIGN

by

Nai-Fung Chen

May 1975



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-75-724

AN ANALYSIS OF SCHEDULING ALGORITHMS IN
MULTIPROCESSOR COMPUTING SYSTEMS

by

Nai-Fung Chen

May 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the Department of Mathematics, the Department of Computer Science, and the National Science Foundation under grant GJ-41538 and was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Mathematics, 1975.

ACKNOWLEDGMENT

I wish to express my sincere thanks to Professor C. L. Liu for his inspiring guidance and constant encouragement during the course of preparing this thesis. I would also like to thank Drs. Andrew and Frances Yao for numerous constructive conversations and discussions on the subject of this thesis as well as on other subjects.

Last but not least, I wish to thank my wife, Ann, for keeping me happy throughout my studies.

TABLE OF CONTENTS

CHAPTER	Page
1. INTRODUCTION.....	1
1.1 Motivation.....	1
1.2 A General Multiprocessing System.....	2
1.3 Optimal Scheduling Problems and NP-Complete Problems.....	6
1.4 Heuristic Scheduling Algorithms.....	8
1.5 A Preview.....	9
2. A SURVEY OF PREVIOUS WORKS.....	10
2.1 Introduction.....	10
2.2 The General Multiprocessing System.....	10
2.2.1 A General Bound.....	10
2.2.2 Some Special Bounds.....	11
2.2.2.1 The Critical Path Algorithm.....	11
2.2.2.2 The Coffman-Graham Algorithm (CGA).....	13
2.3 The Augmented General Multiprocessing Model.....	15
2.4 The Heterogeneous Computing System.....	17
2.5 Conclusion.....	21
3. BOUNDS ON THE CRITICAL PATH SCHEDULING ALGORITHM.....	22
3.1 Introduction.....	22
3.2 Notations.....	23
3.3 Some Preliminary Lemmas.....	24

CHAPTER	Page
3.3.1 Part 1 of the Proof of Theorem 1: $a_1 \leq a_n$ is assumed.....	25
3.3.2 Part 2 of the Proof of Theorem 1: $a_1 > a_n$ is assumed.....	27
3.4 Conclusion of the Proof of Theorem 1.....	41
4. A SUFFICIENT CONDITION FOR THE OPTIMALITY OF THE CRITICAL PATH ALGORITHM.....	46
4.1 Introduction.....	46
4.2 A Generalization of Hu's Theorem.....	46
5. ON A VARIATION OF THE CRITICAL PATH ALGORITHM.....	52
5.1 The Critical Path and Count Algorithm.....	52
5.2 Optimality of the Critical Path and Count Algorithm.....	53
6. ON THE SUCCESSORS ALGORITHM.....	59
6.1 The Successors Algorithm.....	59
6.2 A Bound on the Successor Algorithm.....	59
7. CONCLUSION.....	65
LIST OF REFERENCES.....	66
VITA.....	68

LIST OF FIGURES

Figure	Page
1.1 A directed graph representing a set of jobs.....	4
1.2 A timing diagram representing a schedule.....	5
3.1 A partition of CPS in Type I and Type II blocks.....	31
3.2 An illustration of $X(I)$ and $X(II)$ block of Lemma 3.8..	34
3.3 An example of $\omega_{CPA}/\omega_0 \rightarrow \frac{5}{3}$ for a 4-processor system....	44
4.1 An example of $<$ satisfying hypothesis of Theorem 2....	47
5.1 Example of Algorithm CPCA.....	56
5.2 Example for Definition of X_i	57
6.1 An example for which $\omega_{SA}/\omega_0 = \frac{4}{3}$	64

CHAPTER 1

INTRODUCTION

1.1 Motivation

One of the most significant advantages attributed to multiprocessor systems is the potential decrease in computation time for a large class of problems achievable by parallel programming, that is, the concurrent execution of independent portion of a computational job. This is especially important in real-time application when the results are needed more quickly than they can be provided by single-processor systems. However, it has been known [8] for some time that certain rather general models of multiprocessing systems frequently exhibit behavior which could be termed "anomalous," e.g., an increase in the number of processors of the system or reducing the execution times of some of the jobs or weakening the precedence constraints among the jobs can cause an increase in the time used to complete a set of jobs. In order to fully realize the potential benefits afforded by parallel processing, it becomes important to understand the underlying causes of this behavior and the extent to which the resulting system performance may be degraded. Indeed, how best to exploit the multiple processor organization in order to obtain maximum performance is the primary concern of this thesis.

The performance of an algorithm can be measured in several rather different ways. Two of the most common involve examining the

expected behavior and the worst-case behavior of the algorithm under consideration. Theoretical results regarding expected behavior require assumptions concerning the underlying probability distributions of the parameters involved and historically have been extremely resistant to attack. Fortunately, there are many situations for which worst-case behavior is the appropriate measure in addition to the fact that worst-case behavior does bound expected behavior. It is this latter measure of performance which will be used on the model and algorithms discussed in this thesis. Since it is essential to have on hand the worst examples one can think of before conjecturing and proving bounds on worst-case behavior, numerous such examples will be given throughout the thesis.

1.2 A General Multiprocessing System

We consider the problem of scheduling a set of jobs on a multiprocessor computing system that has a set of identical processors capable of independent operation on independent jobs. Let $P_1, P_2, \dots, P_n$ denote the n identical processors in a multiprocessor computing system. Let $Q = \{J_1, J_2, \dots, J_r\}$ denote a set of jobs to be executed on the computing system. We assume that the execution of a job occupies one and only one processor. Moreover, since the processors are identical, a job can be executed on any one of the processors. Let $\mu(J_i)$ denote the execution time of job J_i , that is, the amount of time it takes to execute J_i on a processor. There is also a partial ordering $<$ specified over Q . It is required that if $J_i < J_j$ then the execution of job J_j cannot begin until the execution of job J_i has been completed. (J_i is said to be a predecessor of

J_j , and J_j is said to be a successor of J_i . J_j is an immediate successor of J_i if there is no J_k such that $J_i < J_k < J_j$.)

Formally, a set of jobs is specified by an ordered triple $(Q, \mu, <)$ where μ is a function from Q to the reals. A set of jobs can be described by a directed graph such as that in Figure 1.1. There is a directed arc from the vertex (job) J_i to the vertex (job) J_j if and only if J_j is an immediate successor of J_i .

By scheduling a set of jobs on a multiprocessor computing system we mean to specify for each job J_i the time interval within which it is to be executed and the processor P_k on which execution will take place. An explicit way to describe a schedule is a timing diagram, also known as the Gantt chart. As an example, the timing diagram of a schedule for the execution of the set of jobs in Figure 1.1 on a two-processor computing system is shown in Figure 1.2.

All schedules considered in this thesis are non-preemptive meaning that once the execution of a job begins on a processor, it will continue until completion. In a given schedule, an idle period of a processor is defined to be a time interval within which the processor is not executing a job (while at least one other processor is executing some job). In a schedule, a processor might be left idle for a period of time either because there is no ready job within that time period or because it is an intentional choice. (A job is said to be ready at a certain time instant if the execution of its predecessors have all been completed at that time.) Clearly, it is never necessary nor beneficial in a schedule to leave all processors idle at the same time. For unit-time jobs an optimal schedule can always be found among schedules with no intentional idle

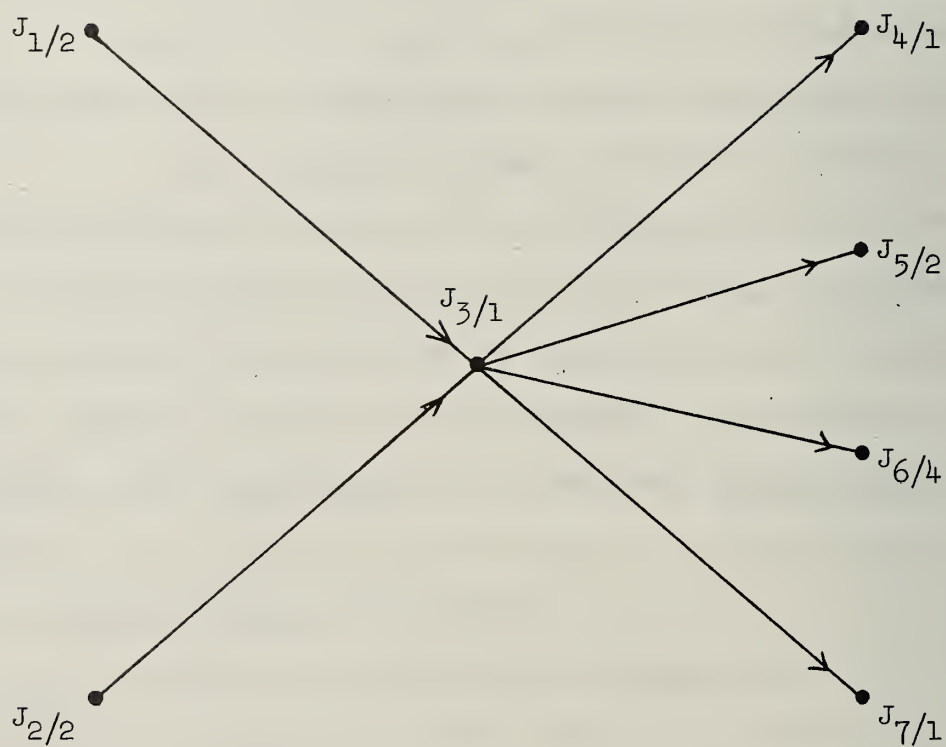


Figure 1.1 A directed graph representing a set of jobs.

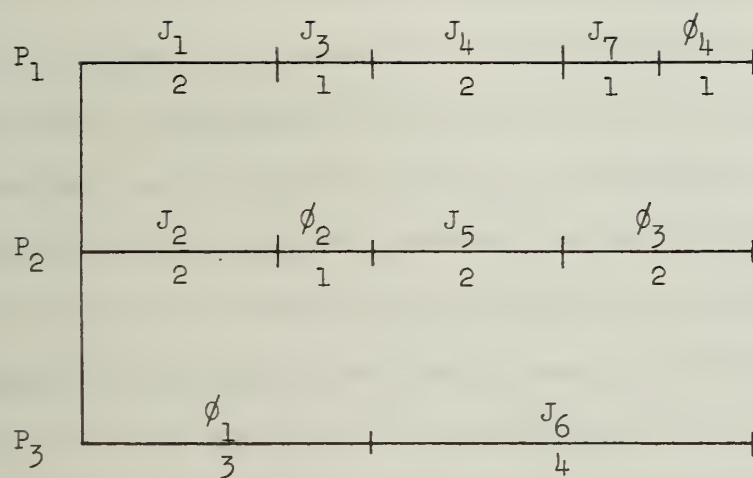


Figure 1.2 A timing diagram representing a schedule.

periods; that is, a processor is left idle for a certain period of time if and only if no job is ready within that period. Thus we need only be concerned with such schedules. In this case, a scheduling algorithm can be specified by merely giving the rules on how jobs are to be chosen for execution at any instant when one or more processor is free. (Of course, the choice is only among jobs that are ready at that instant.) A simple way to spell out the rule is to assign priorities to the jobs so that jobs with higher priorities will be executed instead of jobs with lower priorities when they are competing for processors. (If two or more processors are available, pick the one with smallest index.) Consequently, we call such algorithms priority-driven scheduling algorithms. The usual practice is to list the jobs in decreasing order of their priorities from left to right. Such a list is called a priority list. Whenever a processor is free, the priority list is searched from left to right and the first ready job encountered will be executed.

1.3 Optimal Scheduling Problems and NP-Complete Problems

Given a set of jobs, one of the problems of scheduling is to find an optimal schedule. For a set of n jobs there are only a finite number of schedules and one of them will be an optimal one. However, an exhaustive examination of all the schedules (the number of which exceeds $n!$) requires considerable computation time and will offset the advantage gained by using an optimal schedule. There is a general agreement that an algorithm used to solve a certain problem is considered to be efficient if the number of 'elementary' steps of the algorithm is bounded by a polynomial function of the size of the

problem which in our present case is the number of jobs to be scheduled. Such algorithms are usually called polynomial-time-bounded algorithms or, for conciseness, polynomial algorithms.

Thus the brute force method of examining all the schedules is clearly inefficient since the number of elementary steps involved is exponential in n . There are many classical problems in combinatorics, such as the travelling salesman problem, the Hamiltonian circuit problem, and integer linear programming problem for which no polynomial algorithm producing optimal solutions is known. Recently a class of problems known as nondeterministic polynomial-time complete (or NP-complete) problems which includes all the previously mentioned problems was studied. S. A. Cook [4] and R. M. Karp [11] showed that all problems in this class are equivalent in the following sense: If one problem of this class has a polynomial solution, then all problems in this class have. Since many of these problems are practically important and have been studied by mathematicians and computer scientists for decades and no polynomial solution has been found for even one of them, it is natural to conjecture that no such polynomial algorithms exist. But, in spite of the overwhelming empirical evidence to the contrary, it is still an open question whether NP-complete problems admit of polynomial solution.

J. D. Ullman [14] showed that for the multiprocessing system described in Section 1.2 the problem of determining an optimal algorithm for n -processor systems for all n is NP-complete. More specifically, he showed that the problem of determining an optimal schedule is NP-complete even for the following cases.

(1) All jobs in the given job set have equal execution time. The number of processors in the system is arbitrary.

(2) There are only two processors in the system. The execution time of each job in the job set is either one or two.

Thus the problem of designing an optimal scheduling algorithm is difficult indeed.

Several authors have designed efficient algorithms to produce optimal schedules [3], [5], [10]. Unfortunately, these algorithms are only applicable to some special cases. As a matter of fact, polynomial algorithms that produce optimal schedules are known only for the following special cases:

(1) All jobs have equal execution time and the partial ordering over the jobs is such that either every job has at most one successor or every job has at most one predecessor [10].

(2) All jobs have equal execution time and there are only 2 processors in the computing system [3], [5].

1.4 Heuristic Scheduling Algorithms

In view of the difficulties in practice, one has to use approximate heuristic scheduling algorithms which hopefully yield 'good' schedules in a reasonable amount of computing time. Thus instead of seeking an algorithm which produces optimal schedules, one seeks an approximation algorithm from the set of 'sufficiently efficient' algorithms. Unfortunately, it is usually difficult to evaluate and compare the performance of heuristic algorithms other than by running them on large problem sets with known optimal solutions. A more rigorous approach is to analyze mathematically the performance of such algorithms to determine how closely the constructed

solutions approximate optimal solutions. In this thesis a number of heuristic algorithms are considered. Worst-case performance bounds relative to the optimal solution are determined. Sufficient conditions under which these heuristic algorithms are optimal are given.

Worst-case results on performance, though not as practical as results on average performance, are quite useful, especially because they enable one to guarantee that a particular algorithm will never exceed the optimal solution by more than a known, hopefully small, percentage. Intuitively, one might expect that a mechanism which causes a particular algorithm to have a certain worst-case behavior might also be expected to manifest itself to a certain extent in the average case.

1.5 A Preview

A review of past results on the model described in Section 1.2 as well as other related models is given in Chapter 2. In Chapter 3, bounds on the worst-case performance of the critical path algorithm are derived. A generalization of T. C. Hu's [10] theorem is obtained in Chapter 4. Chapter 5 gives an analysis of an algorithm which is a variation of the critical path algorithm. Finally, the successors algorithm proposed by R. L. Graham [8] is analyzed in Chapter 5.

CHAPTER 2

A SURVEY OF PREVIOUS WORKS

2.1 Introduction

In this chapter previous works on the scheduling problem for the general multiprocessing model described in Section 1.2 are reviewed. Furthermore, works on Garey and Graham's [6] augmented model with multiple resources and Liu and Liu's results [13] on the heterogeneous model will be presented.

2.2 The General Multiprocessing System

2.2.1 A General Bound

For the general multiprocessing model described in Section 1.2, R. L. Graham [8] obtained the following remarkable theorem:

Let $(q, \mu, <)$ be a given set of jobs to be executed on an n -processor system. Let L and L' be two priority lists, and ω and ω' be the corresponding completion times when the jobs are executed according to these lists. Then, for an n -processor system

$$\frac{\omega'}{\omega} \leq \frac{2n-1}{n} .$$

Moreover, the bound is best possible in the sense that the right-hand side cannot be replaced by any smaller function of n .

The significance of this result is as follows:

Although it is desirable to have a priority list such that the

total completion time is minimal, any arbitrary priority list will not lead to an increment of more than 100%. Since searching for an optimal priority list is quite time-consuming--as a matter of fact, there is no general algorithm short of exhaustion--the result in Graham's theorem is indeed comforting.

Graham's theorem can be extended to a more general form.

Let $(Q, \mu, <)$ and $(Q', \mu', <')$ be two sets of jobs such that

- (i) $Q = Q'$,
- (ii) $J_i < J_j \Rightarrow J_i <' J_j$ for any $J_i, J_j \in Q$,
- (iii) $\mu(J) \geq \mu'(J)$ for any $J \in Q$.

Let ω and ω' be the total execution times when these two sets of jobs are executed according to two lists L and L' on an n -processor and an n' -processor system, respectively. Then

$$\frac{\omega'}{\omega} \leq 1 + \frac{n-1}{n'} .$$

2.2.2 Some Special Bounds

The results in the previous theorems say that even if no effort is spent in searching for a priority list, the performance of any arbitrary list will not be "too far off" from that of an optimal one. Another possible attitude toward the scheduling problem would be to spend some effort in searching for a priority list and hope that the performance of the list so obtained is "closer" to that of an optimal one.

2.2.2.1 The Critical Path Algorithm

Let $(Q, \mu, <)$ be a set of jobs. A chain C in (Q) is a sequence of jobs J_i , $i=1, \dots, m$ such that $J_i < J_{i+1}$ for $i=1, \dots, m-1$. J_1 is said to head the chain C . The length of the chain C equals

$\sum_{i=1}^m \mu(J_i)$. Let $C(J)$ be a chain headed by J which is of maximal length among all chains headed by J . The critical path algorithm (CPA) is a priority driven scheduling algorithm such that

(i) If length of $C(J_1)$ is greater than the length of $C(J_2)$, then CPA assigns a higher priority to J_1 than to J_2 , for any 2 jobs J_1, J_2 of $\mathcal{Q}$.

(ii) If length of $C(J_1)$ equals that of $C(J_2)$ then CPA makes an arbitrary assignment of priorities between J_1 and J_2 .

When $\mu(J) = 1$ for all J in $\mathcal{Q}$ then the length of $C(J)$ is usually called the level of J . The CPA is then known as the level algorithm. Hu [10] has shown that the level algorithm is optimal when the partial ordering $<$ is a forest. A generalization of Hu's result appeared in [1] and will be presented in Chapter 4.

Let ω_{CPA} be the length of a critical path schedule and ω_0 be the length of an optimal schedule, for a given set of jobs $\mathcal{Q}$.

In [9] Graham gave examples to show that $\omega_{\text{CPA}}/\omega_0$ could be arbitrarily close to $2 - \frac{1}{n}$ for an n -processor system. One notes that $2 - \frac{1}{n}$ is the bound for ω_L/ω_0 for an arbitrary priority list L in an n -processor system.

However, Graham conjectured in [9] that the bounds for $\omega_{\text{CPA}}/\omega_0$ could be improved when all jobs in $\mathcal{Q}$ are of equal execution time. In Chapter 3 of this thesis this is shown to be the case.

The following results are due to R. L. Graham [8] for the special case in which $<$ is empty, (i.e., all jobs are independent).

Theorem: Let L^* be a priority list obtained by CPA. To be precise, a job will have a higher priority if it has a longer execution time.

Let ω_{CPA} be the corresponding completion time. Let ω_0 be the

completion time for an optimal priority list. Then

$$\frac{\omega_{CPA}}{\omega_0} \leq \frac{4}{3} - \frac{1}{3n}.$$

Moreover, this bound is the best possible.

Theorem: Suppose the k jobs with the longest execution times are picked and the k highest priorities are assigned to them in such a way that they will be executed optimally. Let $L(k)$ be a priority list obtained by appending to such an assignment of priorities arbitrary assignment of priorities to the rest of the jobs. Let $\omega(k)$ be the corresponding completion time. Then

$$\frac{\omega(k)}{\omega_0} \leq \frac{1 - \frac{1}{n}}{1 + \lfloor \frac{k}{n} \rfloor}.$$

The bound is best possible if $k \equiv 0 \pmod{n}$.

2.2.2.2 The Coffman-Graham Algorithm (CGA)

The CGA which appeared in [3] is defined only for job sets $(Q, \mu, <)$ with $\mu(J) = 1$ for all $J \in Q$. Coffman and Graham showed that CGA produces optimal lists when $\mu(J) = 1$ for all J in $(Q, \mu, <)$ and there are only 2 processors in the system. Another algorithm which produces optimal lists under the same two conditions is due to Fujii, Kasami and Ninomiya [5]. Their algorithm is based on a matching algorithm for bipartite graphs of Edmonds.

Let $(Q, \mu, <)$ be a set of n jobs such that $\mu(J) = 1$ for all $J \in Q$. Let $S(J)$ be the set of all immediate successors of J . In Coffman-Graham algorithm, integer labels will be assigned to jobs and $\alpha(J)$ will be used to denote the integer label assigned to J .

Moreover, $N(J)$ will be used to denote the ordered sequence of integers formed by ordering the set $\{\alpha(J') \mid J' \in S(J)\}$ in decreasing order.

The Coffman-Graham (CGA) Algorithm: Suppose $(Q, \mu, <)$ is a set of jobs such that $\mu(J) = 1$ for all J in Q .

1. Choose an arbitrary job J_0 from Q with $S(J_0) = \emptyset$, and define $\alpha(J_0)$ to be 1.

2. Suppose for some $i \leq n$, that labels $1, 2, \dots, i-1$ have been assigned to $i-1$ jobs. Let R be the set of jobs with no unlabelled successors, i.e., for all $J \in R$, $N(J)$ is defined. Let J^* be a job in R such that $N(J^*)$ is lexicographically smaller than $N(J)$ for all J in R . (Break ties at will.) Define $\alpha(J^*)$ to be i .

3. When all jobs have been labelled, construct a priority list $L = (J_n, J_{n-1}, \dots, J_1)$ such that $\alpha(J_i) = i$ for all i , $1 \leq i \leq n$. L will be the priority list used to determine a schedule for $(Q, \mu, <)$.

Given a set $(Q, \mu, <)$ of jobs with $\mu(J) = 1$ for all $J \in Q$ let ω_{CGA} be the total execution time executing Q using the list produced by the CGA algorithm. Let ω_0 be the corresponding time using an optimal list.

Theorem (Lam and Sethi [12]): For an n -processor system and

$(Q, \mu, <)$ a set of jobs with $\mu(J) = 1$ for all $J \in Q$,

$$\frac{\omega_{CGA}}{\omega_0} \leq 2 - \frac{2}{n}.$$

Moreover the above bound is the best possible.

2.3 The Augmented General Multiprocessing Model

Recently Garey and Graham [6] introduced the augmented general multiprocessing model by postulating the existence of a set of "resources" with the property that at no time may the system use more than some predetermined amount of each resource.

Suppose a set of jobs $(Q, \mu, \langle \rangle)$ are executed according to a given schedule. Let $f(t)$ be the set of jobs which are being executed at time t . The restriction that there are n processors can be expressed by requiring $|f(t)| \leq n$ for all t . Assume that a set of resources $R = \{R_1, R_2, \dots, R_s\}$ is given and that these resources have the following properties. The total amount of resource R_i available at any time is 1 (normalized without loss of generality). A job J_k requires the use of $R_i(J_k)$ units of resources R_i at all times during its execution, where $0 \leq R_i(J_k) \leq 1$. At any t , let $r_i(t)$ denote the total amount of resource R_i which is being used at time t . Thus,

$$r_i(t) = \sum_{J_k \in f(t)} R_i(J_k) .$$

In this augmented model, the fundamental constraint is simply this:

$$r_i(t) \leq 1 \text{ for all } t \text{ and } i=1, \dots, s .$$

In other words, at no time can we use more of each kind of resource than is available.

There are essentially three results in [6]. Assume a set $(Q, \mu, \langle \rangle)$ of r jobs with $\mu(J_i) = t_i$ for $i=1, \dots, r$ and a set of resources $R = \{R_1, \dots, R_s\}$, job resource requirements $R_i(J_k)$ and a

positive integer n , the number of processors, are given. For an arbitrary list L , let ω_L be the total execution time for the augmented system of n processors executing Q according to list L . Let ω_0 be the corresponding time for an optimal list.

Theorem A [6]: For $R = \{R_1\}$ and L arbitrary,

$$\frac{\omega_L}{\omega_0} \leq n .$$

Theorem B [6]: For $R = \{R_1, \dots, R_s\}$, $< \text{empty}$, $n \geq r$, and L arbitrary,

$$\frac{\omega_L}{\omega_0} \leq s + 1 .$$

Theorem C [6]: For $R = \{R_1, R_2, \dots, R_s\}$, $< \text{empty}$, $n \geq 2$, and L arbitrary,

$$\frac{\omega_L}{\omega_0} \leq \min\left(\frac{n+1}{2}, s + 2 - \frac{2s+1}{n}\right) .$$

Furthermore, examples have been found to show that each of these results is essentially the best possible.

Thus, the addition of limited resources into the standard model causes an increase in the worst-case behavior bounds, as might be expected.

However, by restricting all $t_i = 1$, A. C. Yao, et al. [7] have proved the following theorems recently.

Theorem D [7]: For $< \text{empty}$, $t_i = 1$ for all i , $m \geq n$ and s , L being arbitrary,

$$\omega_L \leq \left(s + \frac{7}{10}\right)\omega_0 + s + 2 .$$

Furthermore, the coefficient of ω_0 is best possible.

Comparing this result to that of Theorem B one sees that the restriction that all $t_i = 1$ allowed the bound on ω_L/ω_0 to be strengthened from $s+1$ to $s + \frac{7}{10}$. It had been previously shown by A. C. Yao [15] that asymptotically ω_L/ω_0 could not exceed $s + \frac{17}{20}$.

As might be suspected from Theorem A, the restriction that all $t_i = 1$ for general $<$ is not sufficient to prevent the ratio ω_L/ω_0 from blowing up rather badly. The strongest known results are the following:

Theorem E [7]: If $t_i = 1$ for all i , $n \geq r$ and $s, <$, L being arbitrary, then

$$(1) \quad \frac{\omega}{\omega_0} \leq \frac{1}{2} s \omega_0 + \frac{1}{2} s + 1$$

and (2) There exist examples for which

$$\frac{\omega}{\omega_0} \geq \frac{1}{2} s \omega_0 + \frac{1}{2} s + 1 - \frac{2s}{\omega_0}.$$

If the schedule is formed by a critical path algorithm, then the worst-case bound on ω_{CP}/ω_0 improves considerably. In particular, the following result can be proved [7].

Theorem F [7]: If $t_i = 1$ for all i , $n \geq r$ and $s, <$ being arbitrary, then

$$(1) \quad \frac{\omega_{CPA}}{\omega_0} \leq \frac{17}{10} s + 1;$$

(2) For any $\epsilon > 0$, there exist examples for which

$$\frac{\omega_{CPA}}{\omega_0} > \frac{17}{10} s + 1 - \epsilon.$$

2.4 The Heterogeneous Computing System

In the previous models, the multiprocessing computing systems contain identical processors. In this section a more

general model in which different processors have different computation speeds is described and some known results will be presented. This model is a realistic one when one considers the possibilities of replacing one or more of the processors in an existing system by faster processors and of interconnecting different computers in an installation.

By a heterogeneous computing system, one means a multiprocessor system in which processors have different computing speeds. One measures the speed of a processor against that of a "standard" processor whose speed is taken to be 1. The speed of a processor is said to be b if it is b times as fast as a standard processor. Without loss of generality one may assume that $b > 1$. A multiprocessor system which contains n_1 processors of speed b_1 , n_2 processors of speed b_2 , ..., n_k processors of speed b_k is denoted by

$$P = (n_1, n_2, \dots, n_k; b_1, b_2, \dots, b_k).$$

All the theorems that follow are due to Liu and Liu and appeared in [13].

Theorem: Suppose a multiprocessor system

$$P = (n_1, n_2, \dots, n_k; b_1, b_2, \dots, b_k)$$

is given. Let ω and ω' be the completion times of a set of jobs $(Q, \mu, <)$ when it is run on P according to a priority-driven schedule specified an arbitrary priority list L and an arbitrary (not necessarily priority-driven) schedule, respectively. Suppose that $b_1 > b_2 > \dots > b_k$. Then

$$\frac{\omega}{\omega'} \leq \frac{b_1}{b_k} + 1 - \frac{b_1}{\sum_{i=1}^k n_i b_i}.$$

Moreover, the bound is the best possible.

The above theorem can be generalized as follows: Let

$$P' = (n'_1, n'_2, \dots, n'_k; b'_1, b'_2, \dots, b'_k)$$

be another multiprocessing system. Let ω'' be the completion time when $(Q, \mu, <)$ is executed on P' according to an arbitrary (not necessarily priority-driven) schedule. Then

Theorem:

$$\frac{\omega}{\omega''} \leq \frac{b_1}{b_k} + \frac{\sum_{i=1}^{k'} n'_i b'_i}{\sum_{i=1}^k n_i b_i} - \frac{b'_1}{\sum_{i=1}^k n_i b_i}.$$

Moreover, the bound is the best possible.

Another interesting problem is to compare the execution of a set of jobs on two different multiprocessor systems using a priority-driven schedule specified by the same priority list. This is a realistic situation when some of the processors in an existing system were replaced by processors of different speed yet the priorities assigned to the jobs to be executed were not changed. To illustrate the point, the following special case in which a set of jobs (Q, μ) with $<$ empty is to be executed on two multiprocessor systems $P = (n+1; 1)$ and $P' = (1, n; b, 1)$ according to the same priority list L was investigated. Let ω and ω' denote the respective completion times. Then

Theorem:

$$\frac{\omega}{\omega'} \leq \begin{cases} b & \text{if } b \leq 2 \\ \frac{2(n+b)}{n+2} & \text{if } b > 2 \end{cases}$$

Between the two extremes of using an optimally chosen priority list and of using a completely arbitrary priority list, there is the possibility of using priority lists obtained by simple heuristic procedures. Such a possibility offers a good compromise between the performance of the resultant scheduling algorithm and the computational cost of determining the priority list.

Liu and Liu [13] considered the following problem. A set of jobs (q, μ) with $<$ empty is given. Suppose the r longest jobs in q are chosen. Let ω denote the completion time of these jobs when they are executed on

$$P = (n_1, n_2, \dots, n_k; b_1, b_2, \dots, b_k)$$

according to a priority list L_r . Let L denote the priority list obtained by appending to L_r an arbitrary assignment of priorities to the remaining jobs in q . Let ω_r denote the corresponding completion time, and let ω' denote the completion time according to an arbitrary (not necessarily priority-driven) schedule. When $\omega_r > \omega$ and $b_k = 1$, then

Theorem:

$$\frac{\omega_r}{\omega'} \leq 1 + \frac{1}{Q} - \frac{1}{k \sum_{i=1}^k n_i b_i}$$

where

$$Q = \max \left[\min_{1 \leq j \leq k} \left(\left\lceil \frac{r+1}{\sum_{i=1}^k n_i \lceil b_i \rceil} \right\rceil \cdot \frac{\lceil b_j \rceil}{b_j} - \frac{\lceil b_j \rceil - 1}{b_j} \right), \frac{r+1}{\sum_{i=1}^k n_i b_i} \right]$$

Moreover, the bound is best possible when b_i are integers and

$\sum_{i=1}^k n_i b_i$ divides r .

Another simple heuristic for assigning priorities to the jobs in (Q, μ) with $<$ empty is to assign higher priorities to longer jobs. Let ω denote the completion time when (Q, μ) is executed on $P = (1, n; b, 1)$ using a priority assignment according to the lengths of the jobs. Let ω' be the completion time of an arbitrary (not necessarily priority-driven) schedule.

Theorem:

$$\frac{\omega}{\omega'} \leq \frac{2(n+b)}{b+2}, \quad b \leq 2$$

and

$$\frac{\omega}{\omega'} \leq \frac{n+b}{2}, \quad b \geq 2$$

2.5 Conclusion

There is still a lot of room for further investigation in each of the models surveyed in this chapter. In view of a growing number of researchers in this field some significant results are likely to emerge in the next few years.

CHAPTER 3

BOUNDS ON THE CRITICAL PATH SCHEDULING ALGORITHM

3.1 Introduction

The critical path algorithm (CPA) as described in Section 2.2.2.1 is the basis for many of the project planning techniques such as PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method). Therefore, an investigation into its worst-case performance is of both practical and theoretical interest.

Graham [9] showed that $\omega_{\text{CPA}}/\omega_0$ (same notation as in 2.2.2.1) is upper-bounded by $2 - \frac{1}{n}$ for an arbitrary job set $(Q, \mu, <)$ in an n -processor system, and that this bound cannot be improved. However, he conjectured that when Q consists only of unit-time jobs, then the upper-bound for $\omega_{\text{CPA}}/\omega_0$ could be improved to $2 - \frac{2}{n+1}$. The example at the end of this chapter shows that $\omega_{\text{CPA}}/\omega_0$ could be greater than $2 - \frac{2}{n+1}$ and thus defeats Graham's conjecture.

The rest of this chapter is devoted to the proof of the following theorem [2].

Theorem 1: For an n -processor system and a job set $(Q, \mu, <)$ of unit-time jobs

$$\frac{\omega_{\text{CPA}}}{\omega_0} \leq 2 - \frac{1}{n-1} \quad \text{for } n \geq 3$$

and

$$\frac{\omega_{CPA}}{\omega_0} \leq \frac{4}{3} \quad \text{for } n = 2.$$

Moreover, the above bounds are the best possible.

3.2 Notations

For a given set of jobs of unit execution time, let L be a priority list obtained by the critical path algorithm. The schedule obtained according to L shall be referred to as the critical path schedule (CPS). It may be assumed that execution of jobs starts at $t = 0$. Let ω_{CPA} denote the time at which the execution of all jobs is completed according to the CPS and let ω_0 denote the corresponding time according to an optimal schedule. Throughout the paper, we shall refer to an arbitrarily chosen optimal schedule (among all possible optimal schedules) as the optimal schedule (OPS). When a processor is idle from t to $t+1$ it is said to be executing an empty job during this time interval. An empty job will be denoted ϕ . Hereafter a job shall mean a non-empty job unless the contrary is expressly stated. Let $\alpha: \mathcal{Q} \rightarrow \{0, 1, 2, 3, \dots\}$ be a function such that for any two jobs J_1 and J_2 , $\alpha(J_1) > \alpha(J_2)$ if and only if J_1 has higher priority than J_2 according to the list L , $\alpha(\phi) = 0$ for an empty job ϕ and $\alpha(J) > 0$ if J is non-empty. Corresponding to the CPS let $t: \mathcal{Q} \rightarrow \{0, 1, 2, \dots\}$ be the function such that $t(J)$ equals the time at which the execution of J starts. Let $p: \mathcal{Q} \rightarrow \{0, 1, 2, \dots, n\}$ be the function such that $p(J)$ is the index of the processor which executes J . A column of jobs in CPS is the set of jobs executed in the time unit $[t, t+1]$ for some t , $0 \leq t \leq \omega_{CPA} - 1$. A job J_1 is said to be in the column of job J_2

if $t(J_1) = t(J_2)$. A job I is said to be executed to the right (left) of J if $t(I) > t(J)$ ($t(I) < t(J)$). A block in CPS is a set of consecutive columns in CPS. For a job J in Q $l(J)$ denotes the level of J in Q .

3.3 Some Preliminary Lemmas

Lemma 3.1: If $p(J) = 1$ and $t(J) \leq t(J')$, then $\alpha(J) > \alpha(J')$.

Proof: If $t(J) = t(J')$, then according to convention that the higher priority job is assigned to a processor of lower index, we must have $\alpha(J) > \alpha(J')$. For $t(J) < t(J')$, $\alpha(J') > \alpha(J)$ implies that J' is not ready at $t(J)$; that is, a predecessor of J' , K , is being executed at $t(J)$. Now $\alpha(K) > \alpha(J')$ since L is produced by a critical path algorithm. Thus $\alpha(K) > \alpha(J)$. This implies that $p(K)$ should be assigned the value 1 which is a contradiction. ■

Lemma 3.2: If $\alpha(J) < \alpha(K)$ but $t(J) < t(K)$, then there is a job I , satisfying $t(I) = t(J)$ and $p(I) < p(J)$, which precedes K .

Proof: K is not ready at $t(J)$ otherwise K would have been executed at $t(J)$ or earlier since $\alpha(J) < \alpha(K)$. Thus, there must be a predecessor I of K which is being executed at $t(J)$. We thus have $\alpha(I) > \alpha(K) > \alpha(J)$. It follows that $p(I) < p(J)$ by our convention. ■

We show now a construction of a chain in Q :

1. $i := 1$; let U_1 be any job in the rightmost column of the CPS.
2. $i := i + 1$;
3. Search the rightmost column that is to the left of the column containing U_{i-1} (in CPS) for a job V_i (empty or non-empty) such that $\alpha(V_i) < \alpha(U_{i-1})$.
 - (a) If there is no such V_i , go to (5).

(b) If there is such a V_i then, according to Lemma 3.2, there is a job in the column containing V_i which precedes U_{i-1} . Let this job be U_i .

4. Go to (2).

5. Stop.

We shall refer to the chain obtained this way as "the chain" in Q . Note that if a column contains an empty job $\emptyset$, then either $\emptyset$ or some non-empty job in this column (in CPS) will be a V_i in the above construction, since $\alpha(\emptyset) = 0$. Consequently, the chain contains a job in each of the columns in CPS that have empty jobs.

Let ϕ_i be the set of columns in CPS each of which contains i non-empty jobs, for $i=1, \dots, n$. We shall use a_i to denote $|\phi_i|$.

The proof of Theorem 1 is now divided into 2 parts.

3.3.1 Part 1 of the Proof of Theorem 1: $a_1 \leq a_n$ is assumed

Lemma 3.3: Let $a_1, a_2, \dots, a_n$ be non-negative integer such that $a_1 \leq a_n$ and $n \geq 2$. Let $(Q, <)$ be a set of unit-time jobs that has

$a_1 + 2a_2 + 3a_3 + \dots + na_n$ jobs and contains a chain of length $a_1 + a_2 + \dots + a_{n-1}$.

Then the length of an optimal schedule ω_0 is lower bounded by

$(a_1 + a_2 + \dots + a_n) / (2 - \frac{1}{n-1})$ for $n \geq 3$ and by $\frac{3}{4}(a_1 + a_2)$ for $n = 2$.

Proof: There are 3 cases to be considered.

Case (1): $na_n \leq (n-1)a_1 + (n-2)a_2 + \dots + 2a_{n-2} + a_{n-1}$ and $n \geq 3$. Since Q contains a chain of length $a_1 + \dots + a_{n-1}$ $\omega_0 \geq a_1 + \dots + a_{n-1}$.

The inequality

$$(*) \quad \frac{a_1 + \dots + a_{n-1} + a_n}{a_1 + \dots + a_{n-1}} \leq \frac{2n-3}{n-1} = 2 - \frac{1}{n-1}$$

is equivalent to

$$(n-1)a_n \leq (n-2)(a_1 + \dots + a_{n-1})$$

which implies

$$na_n - (a_n - a_1) \leq (n-1)a_1 + (n-2)(a_2 + \dots + a_{n-1})$$

The last inequality is true by the assumption of Case (1) and

$a_1 \leq a_n$. Hence the inequality (*) is established.

Case (2): $na_n > (n-1)a_1 + \dots + 2a_{n-2} + a_{n-1}$ and $n \geq 3$.

Let $d = na_n - ((n-1)a_1 + \dots + 2a_{n-2} + a_{n-1})$. Hence $d > 0$.

There are $na_n + (n-1)a_{n-1} + \dots + a_1$ jobs in Q and hence

$\omega_0 \geq (na_n + (n-1)a_{n-1} + \dots + a_1)/n$. The inequality

$$(**) \quad \frac{a_n + \dots + a_1}{(na_n + \dots + 2a_2 + a_1)/n} \leq \frac{2n-3}{n-1} = 2 - \frac{1}{n-1}.$$

is equivalent to

$$(n-1)(a_n + \dots + a_1) \leq \frac{2n-3}{n} (na_n + \dots + 2a_2 + a_1).$$

Subtract from both sides $(2n-3)(a_n + \dots + a_1)$, we obtain:

$$\frac{2n-3}{n} ((n-1)a_1 + (n-2)a_2 + \dots + a_{n-1}) \leq (n-2)(a_n + a_{n-1} + \dots + a_1).$$

The righthand side of the inequality is greater than

$$na_n - 2a_n + na_n - d - a_1.$$

The lefthand side is equal to

$$\frac{2n-3}{n} (na_n - d).$$

Thus the right-hand side minus the left-hand side of the last inequality is greater than or equal to $a_n - a_1 + (\frac{n-3}{n})d$ which is non-negative since $a_n \geq a_1$ and $d \geq 0$. Thus inequality (**) is proved.

Case (3): $n = 2$.

There are $2a_2 + a_1$ jobs in Q . Hence

$$\omega_0 \geq \frac{2a_2 + a_1}{2}$$

Thus

$$\begin{aligned} \frac{a_2 + a_1}{(2a_2 + a_1)/2} &= \frac{a_2 + a_1}{a_2 + \frac{a_1}{2}} \leq \frac{a_1 + a_1}{\frac{a_1}{2} + a_1} \text{ since } a_1 \leq a_2 \\ &= \frac{4}{3} \end{aligned}$$

Therefore Lemma 3.3 is proved. ■

Now $\omega_{CPA} = a_n + a_{n-1} + \dots + a_1$. Hence

$$\frac{\omega_{CPA}}{\omega_0} \leq 2 - \frac{1}{n-1} \text{ for } n \geq 3$$

and

$$\frac{\omega_{CPA}}{\omega_0} \leq \frac{4}{3} \text{ for } n = 2.$$

Thus Part 1 is done.

3.3.2 Part 2 of the Proof of Theorem 1: $a_1 > a_n$ is assumed

Part 2 of the proof is divided into steps.

Step 1: Definitions related to the chain

1. A job belonging to the chain in CPS is called a chain job. Otherwise, the job is called a non-chain job.
2. A column in CPS is called a chain column if it contains a chain job. Otherwise, the column is called a non-chain column.

3. A job J in CPS is good if
- (i) it is a non-chain job
 - and
 - (ii) the level of J is higher than or equal to that of the chain job (if there is one) immediately to its right, or there is no chain job to the right of it.

4. A non-chain job J in CPS is bad if it is not good.

Note that a chain job is neither good nor bad but a non-chain job is either good or bad.

5. A column is bad if it has 2 to $n-1$ jobs and all the non-chain jobs in the column are bad.

Lemma 3.4: Any job J in a non-chain column in CPS is good.

Proof: Let U_i be the leftmost chain job that is to the right of J . If J is bad then by definition the level of J is smaller than the level of U_i . Hence $\alpha(J) < \alpha(U_i)$ by definition of the critical path algorithm. There must be a job in the same column as J , or to the right of J that precedes U_i which contradicts, by construction of the chain, the assumption that J is in a non-chain column and U_i is the leftmost chain job to the right of J . ■

Step 2: A definition related to the given optimal schedule OPS

1. A special column (SC) in CPS is a column satisfying the conditions:

- (i) it is a bad column,
- and (ii) all the non-chain jobs of that column (which are bad jobs) are executed to the right of the chain job of that column, in the OPS.

2. The set of columns S_1 is defined to be the set which is the union of ϕ_1 and the set of SC's in CPS. A column belonging to S_1 is referred to as an S_1 -column.

Lemma 3.5: Suppose U_i is the chain job in an S_1 -column. If K is a job which is executed to the right of U_i in the CPS, then K is also executed to the right of U_i in the OPS.

Proof: An S_1 -column has at least one empty job by definition. Since CPS is a priority driven schedule, K is either preceded by U_i or by some bad job J in the column containing U_i . But J is executed to the right of U_i , by definition of a S_1 -column, in the OPS. Hence K must be executed to the right of U_i in the OPS. ■

Lemma 3.6: Same hypothesis and notations as Lemma 3.5. All jobs executed to the right of U_i in CPS are of lower level than that of U_i .

Proof: We shall use the notations in Lemma 3.5. The level of J is at least 2 lower than that of U_i by definition of a bad job. Since K is a successor of either U_i or some J and hence $\ell(K) < \ell(U_i)$. ■

Step 3: A partition of CPS into 2 types of blocks

Let $\phi_n = \phi'_n \cup \phi''_n$ where ϕ'_n is the subset of ϕ_n consisting of all the non-chain columns and ϕ''_n the subset consisting of all the chain columns of ϕ_n .

We introduce now a partition of the CPS: A segment is defined to be either (1) an S_1 -column or (2) the columns between two successive S_1 -columns (excluding the S_1 -columns at both ends), or (3) the columns between an S_1 -column and the right or left end of CPS. Thus the CPS is partitioned into segments. A type I block is defined to be a segment containing at least one ϕ'_n -column. A

type II block is defined to be the union of all the segments between two successive type I blocks. Note that the CPS is thus partitioned into type I and type II blocks so that they alternate (see Figure 3.1).

Step 4: A partition of a type II block into sub-blocks

Type II blocks are further classified and partitioned.

Case (1): If the leftmost block in the CPS is a type II block, it is a type IIB sub-block by definition.

Case (2): A type II block consisting of only one column which is not the leftmost column of CPS is defined to be a type IIA sub-block. (Note that this column must then be an S_1 -column.)

Case (3): The type II block does not belong to case (1) and case (2).

A column in a type II block belonging to this case is defined to be the dividing column if it satisfies the following three conditions:

- (1) It is not the leftmost column of the block.
- (2) The chain job in the column has higher level than that of all other jobs in the same column.
- (3) Among all columns in the block which satisfy (1) and (2) the column is the leftmost one.

Clearly, if there is another S_1 -column in the type II block besides the leftmost column then such a column satisfies conditions (1) and (2) and thus the dividing column exists. Therefore, if a type II block is not the rightmost block of the CPS, the dividing column exists for sure. In this case, we define the type IIA sub-block of this type II block to be the block of columns to the left of the dividing column and excluding the dividing

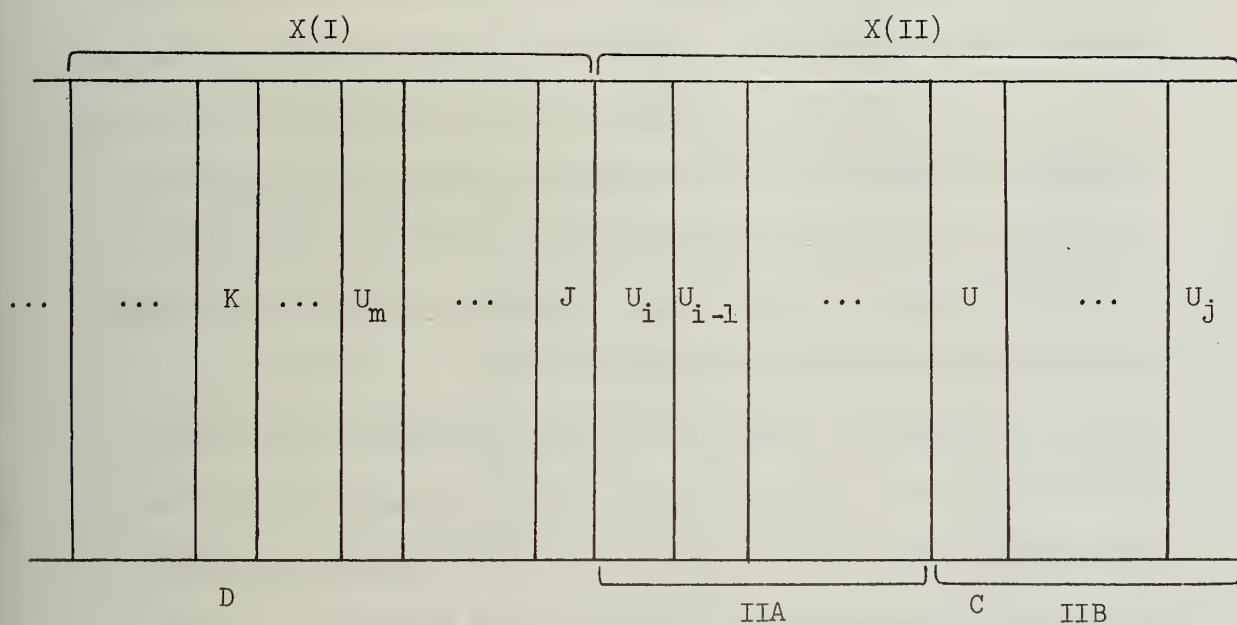


Figure 3.1 A partition of CPS in Type I and Type II blocks.

column, and the type IIB sub-block of this type II block to be the block of columns to the right of the dividing column and including the dividing column. In the case of a type II block that does not contain a dividing column, it is defined to be a type IIA sub-block.

Lemma 3.7: (a) In a column of a type II block the level of any job is lower than or equal to that of the chain job in that column.

(b) If C is a column in a type II block but is not the rightmost column of that block, then each job in C that is of the same level as that of the chain job U in C precedes some job K in the next column on the right of C , and $l(K) = l(V) = l(C) - 1$ where V is the chain job in the column of K .

Proof: The rightmost column of a type II block is either an S_1 -column or is the rightmost column of CPS. In the first case the jobs in the column besides the chain job are either bad jobs or empty jobs by definition of S_1 -column. Bad jobs are at least 2 levels lower than the level of the chain job and empty jobs are of level 0. In the second case all jobs in that column are of level 1 since they have no successors. Thus Lemma 3.7(a) holds for the rightmost column of the type II block.

Suppose, as induction hypothesis, that Lemma 3.7 is true for all columns in the type II block to the right of a certain column C . Let U be the chain job of C . Now U is of higher level than the levels of all chain jobs to its right since U precedes them. In particular $l(U)$ is greater than the level of the chain job of the rightmost column of the type II block. By Lemma 3.6, $l(U)$ is greater than the level of any job executed to the right of this type II block. The levels of jobs in columns on the right of C and within

the type II block are, by induction hypothesis, not higher than the corresponding chain jobs in the respective columns. Thus the levels of all jobs in columns on the right of C and within the type II block are lower than the level of U. Let J be any job in the column of U. Successors of J must all be on the right of C in CPS. Thus the above argument implies the level of any successor of J is lower than that of U. Hence $\ell(U) \geq \ell(J)$. Thus Lemma 3.7(a) is proved. Now if $\ell(U) = \ell(J)$ then J must have a successor K of level $\ell(U)-1$. By Lemma 3.7(a) this successor K could only lie in the column next to and on the right of the column C. The level of this successor K is not higher than the chain job V in its column by Lemma 3.7(a). Thus $\ell(U)-1 \geq \ell(V) \geq \ell(K) = \ell(U)-1$. Hence $\ell(K) = \ell(V) = \ell(U)-1$. ■

Lemma 3.8: Let $X(II)$ be a type II block containing a type IIB sub-block and $X(I)$ a block of (consecutive) columns (not necessarily a type I or type II block) immediately adjacent to the left of $X(II)$ and containing no columns of a type IIB block. Let C be the dividing column of $X(II)$ (hence the leftmost column of the type IIB sub-block of $X(II)$) and U the chain job in C (see Figure 3.2). Then

- (a) If J is a job in the type IIA sub-block of $X(II)$ and is of the same level as the chain job in its column then J precedes U.
- (b) If J is a good job in $X(I)$ then J precedes U.

Proof: If J is a chain job (a) follows from that fact. If J is not a chain job, (a) follows by repeated applications of Lemma 3.7(b) since U is the only job in C whose level is as high as $\ell(U)$.

One notes that $X(I)$ might contain type II block of case (2), i.e., single column type II block and in this case the type II block is a type IIA block.

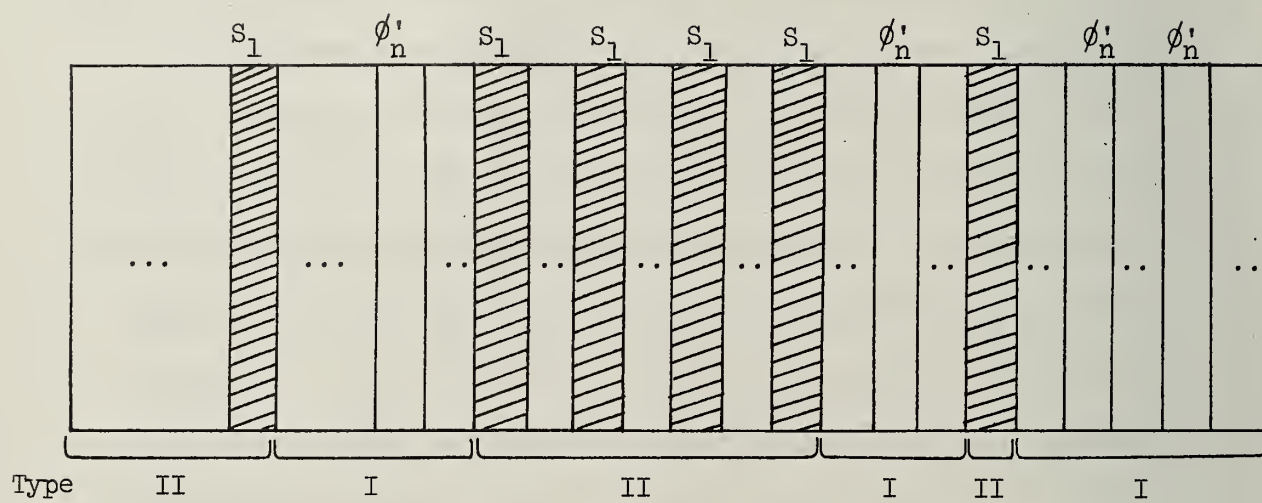


Figure 3.2 An illustration of $X(I)$ and $X(II)$ block of Lemma 3.8.

Let U_i be the chain job in the leftmost column of $X(II)$. This column is an S_1 -column. Let U_{i-1} be the chain job in the column on the right of and next to the column of U_i . U_{i-1} may or may not be U . Then $\ell(U_{i-1}) = \ell(U_i) - 1$ by Lemma 3.7(b).

Let J be a good job in the rightmost column of $X(I)$. By definition of good job $\ell(J) \geq \ell(U_i)$. Successors of J are executed to the right of J in CPS. If J does not precede U_i or any job in the column of U_{i-1} that are of level $\ell(U_{i-1})$, then by Lemma 3.7 and Lemma 3.6 the maximal level of the successors of J is not higher than $\ell(U_i) - 2$. Thus $\ell(J) \leq \ell(U_i) - 1$, contradicting the previous inequality $\ell(J) \geq \ell(U_i)$. Therefore J must precede U_i or some job in the column of U_{i-1} and of level $\ell(U_{i-1})$. By part (a) J precedes U .

Suppose, as induction hypothesis, part (b) is true for all good jobs in $X(I)$ on the right of a certain column D . Let K be a good job in column D . Let U_m be the leftmost chain job executed to the right of K . (So all columns in between D and the column of U_m are non-chain columns and hence contain only good jobs by Lemma 3.4.) By definition of good job, $\ell(U_m) \leq \ell(K)$. If K does not precede any chain job or any good job executed to its right and within the $X(I)$ block, and does not precede U_i or some job in the column of U_{i-1} and of level $\ell(U_{i-1})$, then the successors of K could only consist of the following:

(1) bad jobs B , where $t(U_m) \leq t(B) \leq t(U_i)$ and clearly $\ell(B) \leq \ell(U_m) - 2$, by definition of bad job. (Note: Column of U_i is an S_1 -column.)

(2) jobs J in the column of U_{i-1} and whose level is lower than $\ell(U_{i-1})$. Again $\ell(J) \leq \ell(U_{i-1}) - 1 = \ell(U_i) - 2 \leq \ell(U_m) - 2 \leq \ell(K) - 2$.

(3) jobs J executed to the right of U_{i-1} and in this case $\ell(J) \leq \ell(U_{i-1})-1$, by Lemma 3.7 and Lemma 3.6. Hence $\ell(J) \leq \ell(U_m)-2 \leq \ell(K)-2$.

Thus in any one of these 3 cases the maximal level of the successors of K is not greater than $\ell(K)-2$ which is a contradiction. Therefore K must precede one or more of the following:

(a) a chain job G where $t(K) < t(G) \leq t(U_i)$. Then K precedes U since U is part of the chain.

(b) a good job G where $t(K) < t(G) < t(U_i)$. Then G precedes U by induction hypothesis. Hence K precedes U .

(c) U_i , which is a particular case of case (a).

(d) some job J in the column of U_{i-1} and $\ell(J) = \ell(U_{i-1})$. Then J precedes U by part (a) of this lemma. Thus K precedes U .

One sees that in each case K precedes U , and if K precedes U then K precedes all chain jobs executed to the right of U and in particular those chain jobs in type IIB blocks to the right of $X(I)$. ■

The following corollary easily follows:

Corollary: (a) If J is a job in a type IIA block whose level equals that of the chain job in its column, then J precedes any chain job in a type IIB block which is on the right of the type IIA block, adjacent or otherwise.

(b) If J is a good job in a type I block, then J precedes any chain job in a type IIB block (if there is one) on the right of the type I block, adjacent or otherwise.

Step 5: A reduction of the CPS by the dropping of some jobs from CPS

To find an upperbound for ω_{CPA}/ω_0 it is necessary to find a lower bound for ω_0 . In establishing this lower bound, it is

unnecessary to consider all the jobs. The jobs that will not be included in such a consideration are referred to as jobs which have been dropped.

1. Jobs dropped from columns in type I blocks:

- (a) From a column which is not bad, all bad jobs are dropped.
- (b) A bad column in a type I block is not a special column by definition of a type I block. Hence one or more of the bad jobs in a bad column are not executed to the right of the chain job of the bad column in OPS. From a bad column in a type I block only those bad jobs which are executed to the right of the chain job of the same column in OPS are dropped. Therefore one or more bad jobs and the chain job are not dropped.

2. Jobs dropped from a type II block:

- (a) From a column in a type IIA sub-block, all jobs whose levels are lower than the level of the chain job of the column are dropped.
- (b) From a column in a type IIB sub-block, all jobs except the chain jobs are dropped.

After jobs are dropped from CPS as described in (1) and (2), the resulting schedule is referred to as the reduced critical path schedule (RCPS).

In the future when one speaks of a column or a block of columns one always first specifies the schedule CPS or RCPS. A

column or a block of columns in RCPS is nothing more than the same column or block of columns in CPS minus the jobs dropped with the position of the column or the block of columns along the time-axis remaining the same.

The following lemma is trivial.

Lemma 3.9: (a) From a ϕ'_n -column, no jobs are dropped.

(b) From an S_1 -column, all jobs except the chain job have been dropped.

Proof: (a) ϕ'_n -columns are the non-chain columns which only contain good jobs by Lemma 3.4. ϕ'_n -columns are in type I blocks and they are not bad columns.

(b) Every S_1 -column is in a type II block. An S_1 -column is either a ϕ_1 -column or a bad column. The level of a bad job is at least 2 levels lower than the chain job in its column. Thus all jobs, if there is any, except the chain jobs are dropped from the S_1 -columns. ■

Lemma 3.10: Let U be a (chain) job in a column of a type IIB block in RCPS and J be a job which is not in a type IIB block in RCPS, then $t(J) \neq t(U)$ in OPS.

Proof: $t(U) \neq t(J)$ in RCPS otherwise J belongs to the type IIB block. For any job J in RCPS, $t(J)$ in CPS is equal to $t(J)$ in RCPS. Two cases arise.

Case (1): $t(U) < t(J)$ in RCPS (CPS).

Let U_1 be the chain job in the rightmost column of the type IIB block. This type IIB block is not the rightmost block in CPS for J , which is outside this type IIB block, is executed to its right. Therefore, the rightmost column is an S_1 -column. Now

$t(U_1) < t(J)$ since J does not belong to the type IIB block. By Lemma 3.5, $t(U_1) < t(J)$ in OPS. But U is a chain job hence U precedes U_1 (or $U = U_1$). Thus $t(U) \leq t(U_1)$ in any schedule. Therefore $t(U) < t(J)$ in OPS.

Case (2): $t(J) < t(U)$ in RCPS (CPS).

If J is a chain job then J precedes U . Hence $t(J) \neq t(U)$ in OPS. Hence we may assume J is not a chain job.

There are 3 sub-cases:

(a) J is a job in a type IIB block in RCPS:

J is a job whose level equals that of the chain job in its column otherwise J would have been dropped in Step 5. By the corollary (b) to Lemma 3.8, J precedes U . Thus $t(J) \neq t(U)$ in any schedule and in particular OPS.

(b) J is a good job in a type II block:

Again Corollary (a) to Lemma 3.8 implies J precedes U . Thus $t(J) \neq t(U)$ in OPS.

(c) J is a bad job in a type I block:

Let U_j be the chain job in the column of J in RCPS (CPS). The fact that J is not dropped in Step 5 implies that J is in a bad column such that $t(U_j) \not\leq t(J)$ in OPS. Now U_j precedes U since U_j is on the left of U . Thus $t(U_j) < t(U)$ in OPS. From above $t(J) \leq t(U_j)$. Therefore $t(J) < t(U)$, and hence $t(J) \neq t(U)$. ■

Step 6: Derivation of the upperbounds

Let π_i , $i=1,2,\dots,n$ be the set of columns in RCPS each of which has exactly i jobs. Let π'_n be the subset of π_n consisting of the non-chain columns in RCPS. Thus $\pi'_n = \phi'_n$ since no job in a column in ϕ'_n was dropped in Step 5. Let π''_n be the subset of π_n consisting of

the chain columns of π_n in RCPS. So $\pi_n = \pi'_n \cup \pi''_n$. A column in π_i shall be referred to as a π_1 -column. Thus a type IIB block in RCPS consists completely of π_1 -columns.

Lemma 3.11: (1) Number of π_1 -columns in type I blocks is less than or equal to $|\phi''_n| - |\pi''_n|$.

(2) Number of π_1 -columns in type IIA sub-blocks is less than or equal to $|\pi'_n|$ (note that $|\pi'_n| = |\phi'_n|$).

(3) Number of π_1 -columns is larger than or equal to $|\phi_1|$ plus the number of π_1 -columns in type I blocks.

(4) Number of π_1 -columns in type IIB sub-blocks is larger than or equal to $|\phi_1| - |\phi'_n| \geq |\phi''_n|$.

Proof: (1) According to Lemma 3.4, only columns in the type I blocks in CPS that could possibly become π_1 -columns in RCPS are chain columns in which all non-chain jobs are bad. But 1(b) of Step 5 implies that bad columns in a type I block in CPS could not become π_1 -columns in RCPS. Thus only ϕ''_n -columns can possibly become π_1 -columns. Since the subset π''_n of ϕ''_n -columns remains unchanged and hence only $|\phi''_n| - |\pi''_n|$ could have changed to π_1 -columns.

(2) Recall the dropping in 2(a) of Step 5. By definition of a type IIA sub-block, every column besides the leftmost column of a type IIA sub-block has at least one other job whose level is as high as that of the chain job in the column. Thus none of the columns besides the leftmost column of the type IIA block in CPS could become π_1 -columns. Thus corresponding to each type IIA block in CPS there is only one π_1 -column in RCPS. Now each type I block has at least one ϕ_n -column by definition. So the number of type I blocks is greater than $|\phi'_n|$ which is equal to $|\pi'_n|$. With each type IIA block

one can associate the type I block adjacent to and on the left of the type IIA block. Thus there are no more than $|\phi'_n| = |\pi'_n|$ type IIA blocks and so no more than $|\phi'_n| = |\pi'_n|$ π_1 -columns in type IIA blocks.

(3) All ϕ_1 -columns are in type II blocks and automatically become π_1 -columns in RCPS. Hence (3) follows.

(4) It follows from (1), (2), and (3). ■

3.4 Conclusion of the Proof of Theorem 1

Let π'_1 be the subset of π_1 -columns not in any type IIB block (so in type I and type IIA blocks) whereas π''_1 be the subset of π_1 -columns in type IIB blocks. Then $\pi_1 = \pi'_1 \cup \pi''_1$.

Define $S(\pi_1)$ to be the set of jobs in π_1 -columns in RCPS. Lemma 3.10 shows the execution of the jobs of $S(\pi''_1)$ in OPS is completely independent of the execution of the jobs of $S(\pi'_1 \cup \pi_2 \cup \dots \cup \pi_n)$ in the sense that any job of $S(\pi''_1)$ is not executed at the same time with any job of $S(\pi'_1 \cup \pi_2 \cup \dots \cup \pi_n)$ in OPS. Now there is a chain of length $|\pi'_1| + |\pi_2| + \dots + |\pi_{n-1}| + |\pi''_1|$ in $S(\pi'_1 \cup \pi_2 \cup \dots \cup \pi_n)$. The following cases arise.

Case (1): $n|\pi''_1| \leq (n-1)|\pi'_1| + (n-2)|\pi_2| + \dots + 2|\pi_{n-2}| + |\pi_{n-1}|$ and $n \geq 3$.

By the above remark $\omega_0 \geq (|\pi'_1| + |\pi_2| + \dots + |\pi_{n-1}| + |\pi''_1|) + |\pi''_1|$,

$\omega_{\text{CPA}} = |\phi_1| + \dots + |\phi_n| = |\pi_1| + \dots + |\pi_n|$ since CPS and RCPS have the same length. Thus

$$\frac{\omega_{\text{CPA}}}{\omega_0} \leq \frac{|\pi'_1| + \dots + |\pi'_n| + |\pi''_n| + |\pi''_1|}{|\pi'_1| + \dots + 0 + |\pi''_n| + |\pi''_1|}.$$

The inequality

$$(***) \quad \frac{|\pi_1'| + |\pi_2| + \dots + |\pi_n'| + |\pi_n''| + |\pi_1''|}{|\pi_1'| + |\pi_2| + \dots + 0 + |\pi_n''| + |\pi_1''|} \leq \frac{2n-3}{n-1} = 2 - \frac{1}{n-1}$$

is equivalent to (by similar manipulations as those in Lemma 3.3 of part 1)

$$n|\pi_n'| + (|\pi_1'| - |\pi_n'|) \leq (n-1)|\pi_1'| + (n-2)(|\pi_2| + \dots + |\pi_{n-1}|) + (n-2)(|\pi_1''| + |\pi_n''|).$$

$$\text{Now by assumption } n|\pi_n'| \leq (n-1)|\pi_1'| + (n-2)(|\pi_2| + \dots + |\pi_{n-1}|).$$

$$\begin{aligned} |\pi_1'| - |\pi_n'| &\leq (|\phi_n''| - |\pi_n''|) \leq |\phi_n''| \leq |\pi_1''| \text{ by (1), (2), (4) of Lemma 11} \\ &\leq (n-2)|\pi_1''| \\ &\leq (n-2)(|\pi_1''| + |\pi_n''|). \end{aligned}$$

Thus the above inequality is established and hence (***) is proved.

Case (2): $n|\pi_n'| > (n-1)|\pi_1'| + (n-2)|\pi_2| + \dots + 2|\pi_{n-2}| + |\pi_{n-1}|$ and $n \geq 3$.

$$\text{Let } d_1 = n|\pi_n'| - ((n-1)|\pi_1'| + (n-2)|\pi_2| + \dots + 2|\pi_{n-2}| + |\pi_{n-1}|).$$

Hence $d_1 > 0$. There are $n|\pi_n'| + (n-1)|\pi_{n-1}| + \dots + |\pi_1| + n|\pi_n''|$ jobs in $S(\pi_1' \cup \pi_2 \cup \dots \cup \pi_n' \cup \pi_n'')$, and $S(\pi_1'')$ needs an additional $|\pi_1''|$ units of time. So

$$\begin{aligned} \omega_0 &\geq (n|\pi_n'| + (n-1)|\pi_{n-1}| + \dots + |\pi_1| + n|\pi_n''|) / (n + |\pi_1''|) \\ &= (n|\pi_n'| + (n-1)|\pi_{n-1}| + \dots + |\pi_1|) / (n + |\pi_n''| + |\pi_1''|), \\ \omega_{\text{CPA}} &= |\pi_1'| + |\pi_2| + \dots + |\pi_n'| + |\pi_n''| + |\pi_1''|. \end{aligned}$$

The inequality

$$(\text{****}) \quad \frac{\omega_{\text{CPA}}}{\omega_0} \leq \frac{2n-3}{n-1} = 2 - \frac{1}{n-1}, \quad \text{for } n \geq 3$$

is true if (after similar manipulations as those in Lemma 3.3 of part 1) the following inequality is true

$$0 \leq (|\pi'_n| - |\pi'_1|) + \left(\frac{n-3}{n}\right)d_1 + (n-2)(|\pi''_1| + |\pi''_n|)$$

The inequality $(n-2)(|\pi''_1| + |\pi''_n|) \geq |\pi'_1| - |\pi'_n|$ is proved in case (1).

Since $d_1 > 0$, therefore the last inequality is true. Hence (****) is established.

Case (3): $n = 2$

In $S(\pi'_1 \cup \pi'_2 \cup \pi''_2)$ there are $|\pi'_1| + 2|\pi'_2| + 2|\pi''_2|$ jobs. To execute the jobs in $S(\pi'_1 \cup \pi'_2 \cup \pi''_2)$ OPS need at least $\frac{|\pi'_1|}{2} + |\pi'_2| + |\pi''_2|$. OPS needs an additional $|\pi''_1|$ units of time to execute $S(\pi''_1)$. Thus

$$\omega_0 \geq \frac{|\pi'_1|}{2} + |\pi''_1| + |\pi'_2| + |\pi''_2| ,$$

$$\omega_{CPA} = |\pi'_1| + |\pi''_1| + |\pi'_2| + |\pi''_2| .$$

So $\frac{\omega_{CPA}}{\omega_0} \leq \frac{4}{3}$ is true if

$$|\pi'_1| \leq |\pi'_2| + |\pi''_2| + |\pi''_1|$$

is true. But $|\pi'_1| - |\pi'_2| \leq (|\phi''_2| - |\pi''_2|)$ by (1) and (2) of Lemma 3.11,

$|\pi''_1| \geq |\phi''_2|$ by (4) of Lemma 3.11. So the last inequality is

established. Therefore part 2 of the proof of Theorem 1 is completed.

The partial ordering in Figure 3.3 is an example which shows the bounds in Theorem 1 are exact. If CPA assigns a lower priority to the rightmost job of every level than that of all other jobs of

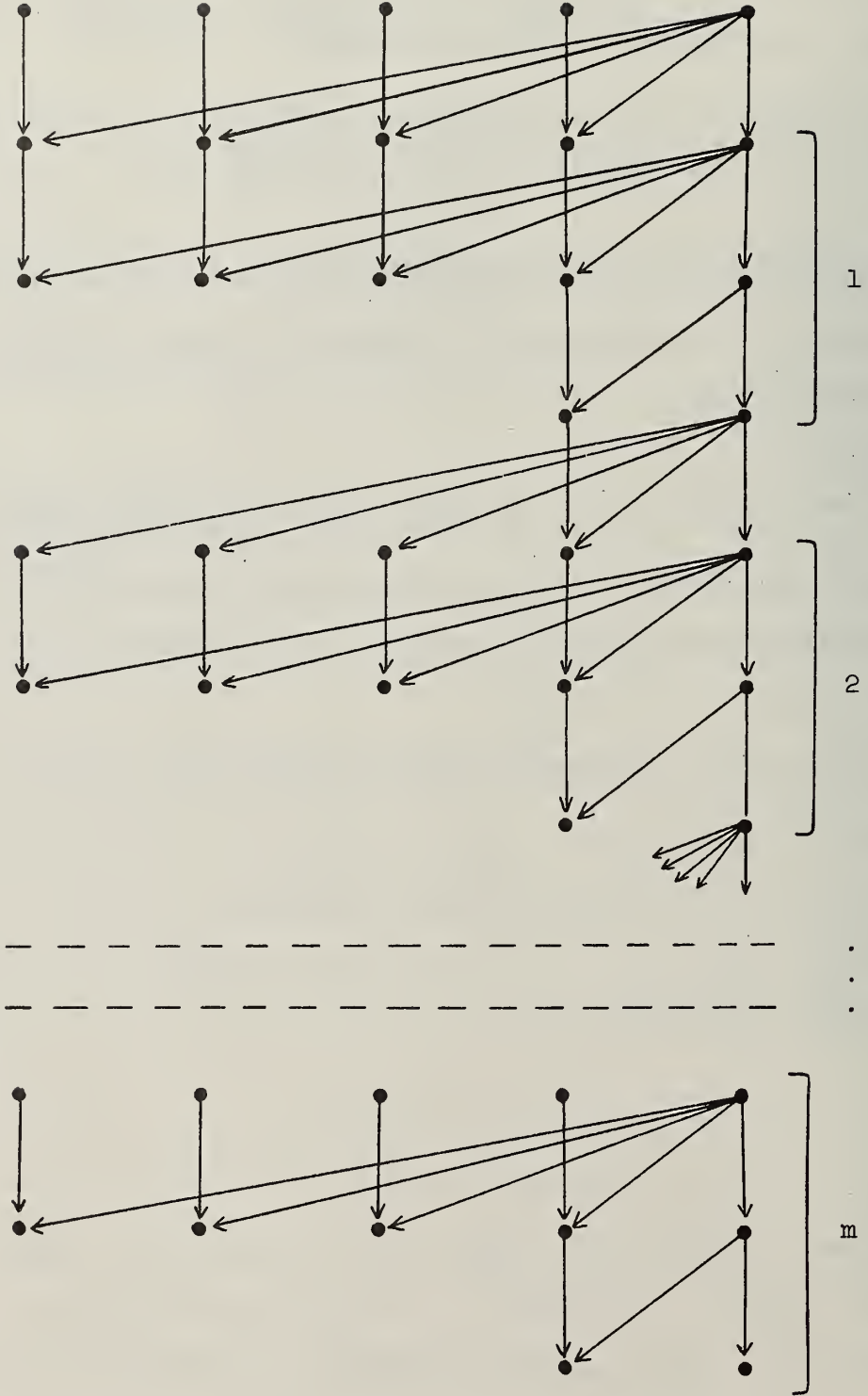


Figure 3.3 An example of $\omega_{CPA}/\omega_0 \rightarrow \frac{5}{3}$ for a 4-processor system.

the same level then the length of CPS equals $7+5(m-1)$ for a 4-processors system. The length of an OPS can be easily shown to be $5+3(m-1)$. Thus $\omega_{CFA}/\omega_0 \rightarrow \frac{5}{3}$ as $m \rightarrow \infty$. The generalization of the above example to n -processors systems for any $n \geq 2$ is obvious. ■

CHAPTER 4

A SUFFICIENT CONDITION FOR THE OPTIMALITY OF
THE CRITICAL PATH ALGORITHM4.1 Introduction

In [10] T. C. Hu proved that for an n -processor system and a job set $(Q, \mu, <)$ with $\mu(J) = 1$ for all $J \in Q$ and $<$ being a forest the critical path algorithm produces an optimal schedule. $<$ is defined to be a forest if every job has at most one successor. Hu's proof is far from simple. In this chapter a simple proof of a generalization of Hu's theorem will be presented.

4.2 A Generalization of Hu's Theorem

For a given set of jobs $(Q, \mu, <)$ where $\mu(J) = 1$ for all $J \in Q$ let $\ell(J)$ be the level of a job J . Let $h = \max_{J \in Q} \{\ell(J) \mid J \in Q\}$. Let N_i be the number of jobs in Q of level i for $i=1, \dots, h$. Two jobs are defined to be independent if there is no precedence relation between them.

Theorem 2: For an n -processor system and a set of jobs $(Q, \mu, <)$, with all $\mu(J) = 1$, if $<$ satisfies the condition that the number of independent jobs in the union of the sets of immediate successors of any n jobs is at most n then the CPA produces an optimal schedule.

Clearly if $<$ is a tree or forest of jobs of equal execution time, then it satisfies the condition in Theorem 2. Thus Theorem 2 is a generalization of Hu's result [10].

Figure 4.1 represents a partial ordering which is not a tree but satisfies the hypothesis of Theorem 2 in the case of 2-processors.



Figure 4.1 An example of $<$ satisfying hypothesis of Theorem 2.

We shall establish 3 lemmas which are needed in the proof of Theorem 2.

Lemma 4.1: Let $(Q, \mu, <)$ be a set of jobs with $\mu(J) = 1$ for all $J \in Q$.

A necessary condition for completing the execution of the jobs in time t , where t is an integer, is

$$N_h + N_{h-1} + \dots + N_1 \leq n(t-i+1) \quad (4-1)$$

for $i=h, h-1, \dots, 1$, where n is the number of processors in the computing system.

Proof: If we are to complete the execution of all jobs at time t , we must complete the execution of all the jobs at levels $h, h-1, \dots, i$

by $t-i+1$. Because if this is not the case, there will be a chain of i jobs to be executed in the remaining $i-1$ time units, which is an impossibility. Since the computing system can execute at most $n(t-i+1)$ jobs in $t-i+1$ units of time, the inequalities in (4-1) follow immediately.

For an arbitrary set of jobs Q , there certainly exists a t which satisfies (4-1). Let $t(Q)$ be the smallest of all such t 's which satisfy (4-1).

The following lemma is trivial.

Lemma 4.2: Under the hypothesis of Theorem 2, m jobs, with $m \leq n$, have at most n independent jobs in the union of their sets of immediate successors.

Lemma 4.3: Under the hypothesis of Theorem 2, if m jobs, with $m \leq n$, $A_1, \dots, A_m$ have n independent jobs $B_1, \dots, B_n$ in the union of their sets of immediate successors, then an immediate successor S of any A_i , $i=1, \dots, m$, is either (1) some B_j or (2) a successor of some B_j or (3) a predecessor of some B_j , $j=1, \dots, n$. Furthermore, if $\{A_1, \dots, A_m\}$ are executed at time u and $\{B_1, \dots, B_n\}$ are executed at time $u+1$ in a certain schedule, then a successor of any A_i , $i=1, \dots, m$, is either some B_j or a successor of some B_j , $j=1, \dots, n$.

Proof: If S is some B_j , we are done. If $S \notin \{B_1, \dots, B_n\}$, then $\{B_1, \dots, B_n, S\}$ is not an independent set of hypothesis of Theorem 2. Hence S either succeeds some B_j or precedes some B_j .

If T is a successor of an A_i , then T is either an immediate successor or the successor of an immediate successor of some A_i . By the argument in the last paragraph T is either some B_j or a successor

of some B_j . The added time condition excludes the possibility that T is a predecessor of some B_j . ■

Proof of Theorem 2: We shall use the terminologies of Lemma 4.1.

Let $t = t(Q)$ in (4-1) where Q is the set of jobs to be executed which satisfies the hypothesis of the theorem. We shall explicitly construct a schedule that executes all the jobs in time t , and this schedule is the same as the one given by the CPA. Let

$$L = (J_1 J_2 J_3 J_4 \dots)$$

be the priority list given by CPA. By definition the first N_h jobs are of level h followed by N_{h-1} jobs of level $(h-1)$ and so on.

Suppose we have t columns of boxes indexed from 1 to t and each column contains n boxes indexed from 1 to n . Suppose we regard the jobs as physical objects and we are going to distribute the jobs among the boxes assigning one job to one box according to the following procedure.

- (1) Initialize $k = 1$.
- (2) Assign J_k to the column with the smallest index which can accommodate J_k (A column can accommodate a job J if that column at that moment contains less than n jobs and all predecessors of J are in columns with smaller index than that of the column) and to the box with the smallest index which is not yet filled.
- (3) Increment k , $k=k+1$.
- (4) If $k < |P|$, GO TO (2).
- (5) STOP.

Thus the jobs are assigned in order of their priorities.

(*) We claim that after step (2) is executed on J_{k_i} where J_{k_i} is the lowest priority job of level i , all jobs which have been assigned at this point are in columns with indices smaller than or equal to $(t-i+1)$, $i=h, h-1, \dots, 1$.

(**) For $i=h$, the claim is true by virtue of the inequalities (4-1). Assume it is true for all i , $h \geq i > j$. Suppose, for contradiction, that in the process of assigning jobs of level j we encounter for the first time that a certain job J of level j cannot be assigned to a column with index smaller than or equal to $(t-j+1)$. Two possibilities arise. Either each column with index smaller than or equal to $(t-j+1)$ already contains n jobs or there are some columns in that range which contain less than n jobs but cannot accommodate J because of some precedence relations. In view of the inequalities (4-1) only the second possibility could arise. By induction assumptions jobs of level $j+1$ or higher are in columns 1 to $(t-(j+1)+1)$ and so column $(t-j+1)$ is filled with jobs of level j . Let $r(< t-j+1)$ be the largest among indices of columns preceding column $(t-j+1)$ which are not filled at the time of assigning J . That is to say all columns succeeding column r up to column $(t-j+1)$ are filled at this time.

(***) We assert that each column, s , $r < s \leq t-j+1$, is a set of independent jobs in the union of the sets of immediate successors of the jobs in the immediately preceding column. This assertion is clearly true of the column $r+1$. For, each job in column $r+1$ is necessarily a successor of some job in column r (and hence an immediate successor) otherwise it should have been assigned to column r in Step (2). Assume the assertion is true up to column s .

Let J' be any job in column $s+1$. Since every job in columns ranging from $r+1$ to s is a successor of some job in column r by induction assumption of the assertion, J' , too, must be a successor of some job in column r . Otherwise J' is not a successor of any job in columns from r to s and so should have been assigned to column r in Step (2). By Lemma 4.3 J' is a successor of some job in column $r+1$. Applying Lemma 4.3 repeatedly we arrive at that J' is a successor of some job in column s and hence an immediate successor. Thus the assertion of paragraph (***) is proved.

Since J cannot be assigned to column r so J is a successor of some job in columns r to $(5-j+1)$ and hence a successor of some job in column r in any case. Applying Lemma 4.3 repeatedly, we arrive at that J is a successor of some job in column $(t-k+1)$. But column $(t-j+1)$ consists of jobs of level j and J is also of level j and hence must be independent. This contradiction completes the induction of paragraph (**) and proves the claim in paragraph (*).

Consequently, all jobs are assigned to boxes in columns from 1 to $t = t(Q)$. Corresponding to this assignment is the following job schedule. A job in box i of column j will be executed on the i^{th} processor at time unit j . It is easily seen that the above schedule is the same as the priority-driven schedule given by the priority list L which is produced by CPA.

Since the above schedule completes the execution of all the jobs in time $t = t(Q)$, it is an optimal schedule by Lemma 4.1. ■

CHAPTER 5

ON A VARIATION OF THE CRITICAL PATH ALGORITHM

5.1 The Critical Path and Count Algorithm

In this chapter a variation of the critical path algorithm, the Critical Path and Count Algorithm (CPCA), defined only for job sets $(Q, \mu, <)$ with $\mu(J) = 1$ for all $J \in Q$, is introduced. CPCA will be shown to produce optimal schedules in 2-processor systems for any job set $(Q, \mu, <)$ where $\mu(J) = 1$ for all $J \in Q$.

Let $(Q, \mu, <)$ be a set of jobs with $\mu(J) = 1$ for all $J \in Q$. Suppose jobs of a certain level i have been assigned priorities in some way. Subsets of jobs of level i will be assigned priorities in the following way. Let S_1 and S_2 be two distinct subsets of level i . S_1 will be assigned a higher priority than S_2 if any one of the following is satisfied:

$$(1) \quad |S_1| > |S_2|$$

(2) $|S_1| = |S_2| = m$ (say). Suppose L_1 and L_2 are the lists of jobs in S_1 and S_2 in descending order of priorities. There exists an n , $0 \leq n \leq m-1$, such that the first n jobs in both lists coincide and the $(n+1)^{\text{th}}$ job in L_1 has higher priority than the $(n+1)^{\text{th}}$ job in L_2 .

Remark: Two jobs or two subsets of a level have the same priority if they are identical.

The CPCA is an algorithm such that a job of higher level will be assigned higher priority than a job of lower level. CPCA will be defined once the priorities of jobs within a level are specified. This shall be done inductively as follows:

(1) Assign priorities to jobs of level 1 arbitrarily.

(2) Assume priorities have been assigned to jobs of level i , for all i , $1 < i < n$. Let I and J be two jobs of level n . Let $I(i)$ and $J(i)$ be the set of successors of I and J that are in level i respectively, $1 \leq i < n$. Define

$$L(I) = (I(n-1), I(n-2), \dots, I(1))$$

$$L(J) = (J(n-1), J(n-2), \dots, J(1)) .$$

A higher priority will be assigned to I than J if there exists an m , $0 \leq m < n-1$, such that the first m sets of both lists coincide and the $(m+1)^{\text{th}}$ set of $L(I)$ has higher priority than the $(m+1)^{\text{th}}$ set of $L(J)$. If such an m does not exist then the two lists coincide and we make an arbitrary choice of assignment of priorities between I and J .

5.2 Optimality of the Critical Path and Count Algorithm

For a 2-processor system CPCA will be shown to produce optimal schedules.

Theorem 3: For a 2-processor system and a set $(Q, \mu, <)$ of jobs where $\mu(J) = 1$ for all $J \in Q$

$$\frac{\omega_{\text{CPCA}}}{\omega_0} = 1 ,$$

where ω_{CPCA} and ω_0 are the completion times using respectively a schedule produced by CPCA and an optimal schedule respectively.

The notations in Section 3.2 and Lemma 3.1 of Section 3.3 of Chapter 3 will be used in the proof of Theorem 3. Of course the function α is defined using CPCA instead of CPA and the functions t and p are defined using a schedule produced by CPCA.

Proof of Theorem 3: Suppose Q is executed using a schedule produced by CPCA. Jobs V_i and W_i are defined recursively as follows:

(1) V_0 is defined to be the job executed by P_1 satisfying $t(V_0) = \omega_{\text{CPCA}}^{-1}$ (i.e., V_0 is the last job to be executed by P_1). Similarly W_0 is defined to be the (possibly empty) job executed by P_2 with $t(W_0) = \omega_{\text{CPCA}}^{-1}$.

(2) In general, for $k \geq 1$, W_k is defined to be the (possibly empty) job J for which $\alpha(J) < \alpha(V_{k-1})$, $t(J) < t(V_{k-1})$ and $t(J)$ is maximal. It follows from Lemma 3.1 that W_k must be executed by P_2 . V_k is defined to be the job executed by P_1 satisfying $t(V_k) = t(W_k)$.

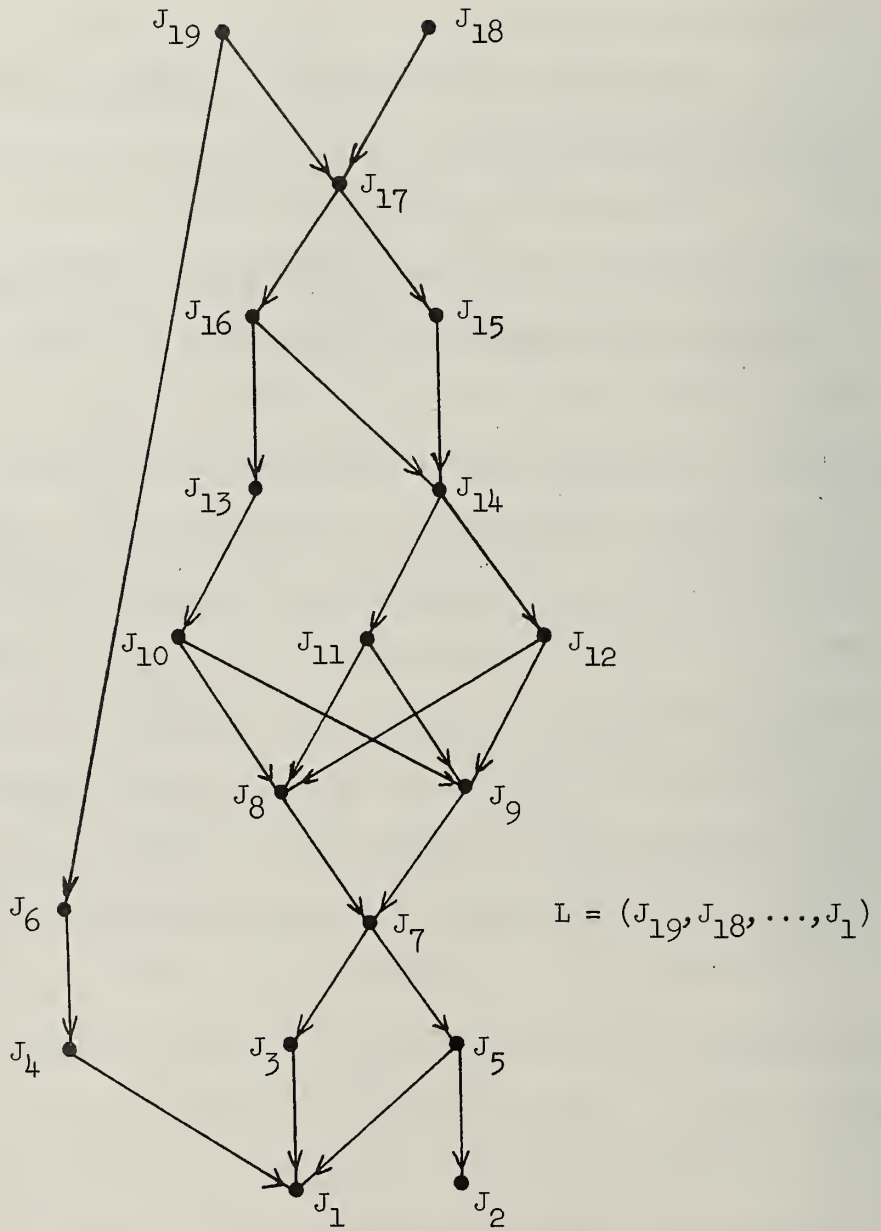
If W_1 does not exist then no processor is idle before time $\omega_{\text{CPCA}}^{-1}$ and ω_{CPCA} is clearly equal to ω_0 . Hence, it may be assumed that W_1 (and therefore V_1) exists. Suppose W_i 's exist for $0 \leq i \leq m$. X_i is defined to be the set of jobs J satisfying $t(V_{i+1}) < t(J) \leq t(V_i)$ but with $J \neq W_i$, $1 \leq i \leq m$. Since V_{m+1} does not exist then X_m is the set of jobs J with $t(J) \leq t(V_m)$ and $J \neq W_m$.

The cardinality of each X_k is odd. Let $|X_k| = 2n_k - 1$ for a positive integer n_k , $0 \leq k \leq m$. An example illustrating the definition of X_i 's is given in Figure 5.2 which is a reproduction

of the timing diagram in Figure 5.1 with boxes drawn to isolate the sets X_i 's.

The heart of the proof of the theorem is contained in the following assertion:

(*) $J \in X_k$ and $J' \in X_{k+1}$ imply $J' < J$ for $0 \leq k \leq m$. For $k = 0$ there is nothing to be proved. Assume, for induction, that (*) is true for all k less than i for some i , $1 \leq i \leq m-1$. Now $\alpha(W_{i+1}) < \alpha(J)$ for all $J \in X_i$ by definition of W_{i+1} . But $t(W_{i+1}) < t(J)$ for all $J \in X_i$. Therefore J is not ready at $t(W_{i+1})$ for all $J \in X_i$. This implies $V_{i+1} < J$ for all $J \in X_i$. Suppose, for induction, that $I' < J$ for all $J \in X_i$ for each I' satisfying $I' \in X_{i+1}$ and $t(I') > t(V_{i+1}) - r$ for some r , $0 < r \leq n_i - 1$. Let I be a job of X_{i+1} such that $t(I) = t(V_{i+1}) - r$. If $I < I'$ for some $I' \in X_{i+1}$ and $t(I) < t(I')$ then $I < J$ for all $J \in X_i$ by the induction hypothesis above. Hence it can be assumed that I precedes no jobs in X_{i+1} . Suppose, for contradiction, that I does not precede some job K in X_i . Now $\alpha(K) > \alpha(V_i)$ by definition of X_i and $\alpha(V_i) > \alpha(W_{i+1})$ by definition of W_{i+1} . Therefore $\alpha(K) > \alpha(W_{i+1})$. Similarly $\alpha(I) > \alpha(V_{i+1})$. All successors of I , with the possible exception of W_{i+1} and of those in X_i , are of lower level than that of V_i , by the first induction hypothesis and the definition of W_j 's and hence that of K . The fact that $\alpha(K) > \alpha(W_{i+1})$ implies that the set of successors of V_{i+1} at the level $\ell(K)$ has higher priority than the set of successors of I at the same level. Successors of I of level higher than $\ell(K)$ are contained in X_i which is contained in the set of successors of V_{i+1} . According to the definition of CPCA $\alpha(V_{i+1}) > \alpha(I)$ which is a contradiction. Thus $I < J$ for all $J \in X_i$. This completes the induction and proves (*).



N.B. Jobs have been indexed so that the index of a job is equal to the label assigned by Algorithm CPCA, i.e., $\alpha(J_j) = j$.

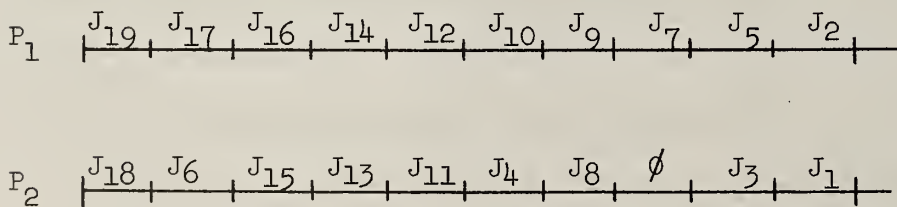
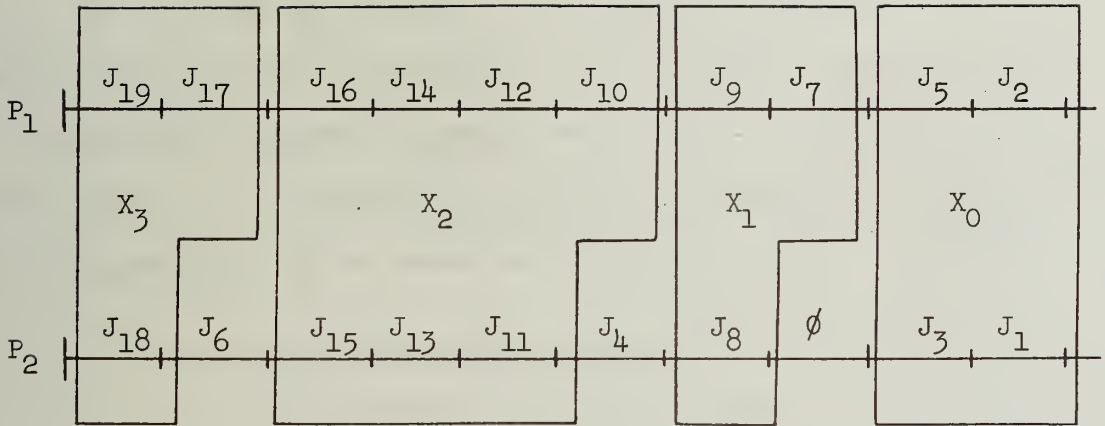


Figure 5.1 Example of Algorithm CPCA.



Note that job indices are chosen to correspond to labels, i.e., $\alpha(J_j) = j$.

Figure 5.2 Example for Definition of X_i .

According to (*) all jobs in X_{k+1} have to be completed before any job in X_k can be started in any schedule. X_{k+1} has $2n_{k+1}-1$ jobs and will require at least n_{k+1} units of time. Thus to complete Q requires at least $\sum_{k=0}^m n_k$ units of time in any schedule. Since $\omega_{CPCA} = \sum_{k=0}^m n_k$ the schedule produced by CPCA is optimal. ■

For $(Q, \mu, <)$ with $\mu(J) = 1$ for all $J \in Q$ CPCA and the Coffman-Graham algorithm are two variations of the CPA which assign higher priorities to higher level jobs. It would be interesting to know if there exists an efficient algorithm which assigns higher priorities to higher level jobs and which is the best among the set of all such algorithms.

CHAPTER 6

ON THE SUCCESSORS ALGORITHM

6.1 The Successors Algorithm

Let $(Q, \mu, \prec)$ be a set of jobs with $\mu(J) = 1$ for all $J \in Q$. The Successor Algorithm (SA) assigns a higher priority to a job with more successors than a job with less successors. If two jobs have the same number of successors then SA breaks tie between them arbitrarily.

6.2 A Bound on the Successor Algorithm

In this chapter the following theorem will be proved.

Theorem 4: For a 2-processor system and a job set $(Q, \mu, \prec)$ with $\mu(J) = 1$ for all $J \in Q$

$$\frac{\omega_{SA}}{\omega_0} \leq \frac{4}{3},$$

where ω_{SA} and ω_0 are the lengths of a schedule produced by the SA and an optimal schedule respectively. Moreover, the bound is the best possible.

Proof: The notations of Section 3.2 and Lemma 3.1 of Section 3.3 will be used. The function α is defined by SA and the functions t and p are defined using a schedule produced by SA. SS will be used to represent a schedule produced by the SA. OPS is an arbitrarily chosen optimal schedule (chosen from the set of all possible optimal schedules). Let ϕ_1 be the set of columns in SS each of which has

only one job and ϕ_2 be the set of columns in SS each of which containing two jobs. Thus $\omega_{SA} = |\phi_1| + |\phi_2|$. A column in ϕ_i will be referred to as a ϕ_i -column for $i=1,2$.

The proof of Theorem 4 is now divided into two parts.

Part 1: $|\phi_1| \leq |\phi_2|$ is assumed

There are $|\phi_1| + 2|\phi_2|$ non-empty jobs in $\mathcal{J}$. Thus

$$\omega_0 \geq (|\phi_1| + 2|\phi_2|)/2.$$

Therefore

$$\begin{aligned} \frac{\omega_{SA}}{\omega_0} &\leq \frac{|\phi_1| + |\phi_2|}{(|\phi_1| + 2|\phi_2|)/2} \\ &\leq \frac{|\phi_2| + |\phi_2|}{(|\phi_2| + 2|\phi_2|)/2} \quad \text{since } |\phi_1| \leq |\phi_2|, \\ &= \frac{4}{3}. \end{aligned}$$

Part 2: $|\phi_1| > |\phi_2|$ is assumed

The heart of the proof lies in the fact that OPS has at least $|\phi_1| - |\phi_2|$ empty jobs.

Jobs V_i 's, W_i 's and sets of jobs X_i 's are defined in exactly the same way as in the proof of Theorem 3. The following properties about V_i 's, W_i 's and X_i 's are true independent of the algorithm used to produce the schedule.

- (1) $V_i < V_j$ for all $0 \leq j < i \leq m$.
- (2) $V_i < J$ for all $J \in X_j$ with $0 \leq j < i \leq m$.

- (3) $\alpha(W_{i+1}) < \alpha(V_i) < \alpha(J)$ for all $J \in X_i$ with
 $0 \leq i \leq m$.

By definition of the V_i 's and W_i 's each ϕ_1 -column contains a V_j and a $W_j = \phi$ since $\alpha(\phi) = 0$. Let ϕ_2'' be the subset of ϕ_2 consisting of all ϕ_2 -columns each of which contains a V_k (and hence a W_k too). Let ϕ_2' be $\phi_2 - \phi_2''$. Thus $\phi_2 = \phi_2' \cup \phi_2''$. It can be seen that ϕ_2' -columns are contained in X_i 's. An X_i consists of some ϕ_2' -columns and the job V_i . Since the V_i 's form a chain (property (1)) there is a chain of length $|\phi_2''| + |\phi_1|$ in Q .

The following assertion will be proved: For any J in X_i where $J \neq V_i$, J precedes V_s where V_s is the (only) job in the second ϕ_1 -column on the right of V_i . Equivalently this assertion says that each job in a ϕ_2' -column precedes the job V_s in the second ϕ_1 -column on the right of the ϕ_2' -column.

The above assertion will be proved by induction. Let $J \in X_i$ and $J \neq V_i$. Suppose, as induction hypothesis, that all jobs in X_i different from V_i and executed to the right of J satisfy the conclusion of the assertion. If J precedes any one of these jobs then J precedes V_s by induction hypothesis. Hence it can be assumed that J precedes no jobs in X_i . It can also be assumed that J precedes no V_j executed to the left of V_s for otherwise $J < V_j < V_s$ and the assertion is proved. Let V_f be the job in the first ϕ_1 -column on the right of V_i . Then V_i precedes the following:

- (a) All V_j , for all $j < i$. This means V_i precedes all V_j executed to its right (property (1)).
- (b) All jobs in X_j , for all $j < i$ (property (2)).

(c) All jobs executed to the right of V_f . This follows from the fact that V_f precedes all jobs executed to its right and $V_i < V_f$.

Now $\alpha(V_i) < \alpha(J)$ according to property (3). Hence J has at least as many successors as V_i . If J does not precede V_s then, by a simple counting on the timing diagram, V_i will have more successors than J which is a contradiction. Thus the assertion is established.

The assertion has the following implication: If J is in a ϕ'_2 -column and V_k is in a ϕ_1 -column, then either $J < V_k$ or $V_k < J$ unless V_k is in the first ϕ_1 -column on the right of the ϕ'_2 -column containing J . Now there are two jobs in each ϕ'_2 -column. Thus the following important conclusion can be drawn: If J_1 and J_2 are the two jobs in a ϕ'_2 -column then either J_1 or J_2 is not executed simultaneously with any V_k where V_k is the job of any ϕ_1 -column in the arbitrarily chosen OPS. Thus at most half of the jobs in ϕ'_2 -columns can be executed with V_k 's in ϕ_1 -columns in OPS. The only other jobs of Q that can be simultaneously executed with V_k 's in ϕ_1 -columns are the W_1 's and there are only $|\phi''_2|$ of them. Now there are $|\phi_1|$ V_k 's in ϕ_1 -columns and hence there are at least $|\phi_1| - \frac{1}{2} \cdot 2|\phi'_2| - |\phi''_2|$ V_k 's in ϕ_1 -columns which are not executed simultaneously with any other job of Q in OPS. Therefore there are at least $|\phi_1| - |\phi_2|$ empty jobs in OPS. Hence

$$\omega_0 \geq (|\phi_1| + 2|\phi_2| + |\phi_1| - |\phi_2|)/2 = |\phi_1| + |\phi_2|/2.$$

Therefore

$$\begin{aligned}
\frac{\omega_{SA}}{\omega_0} &\leq \frac{|\phi_1| + |\phi_2|}{(|\phi_1| + |\phi_2|)/2} \\
&\leq \frac{|\phi_1| + |\phi_1|}{(|\phi_1| + |\phi_1|)/2} \quad \text{since } |\phi_2| < |\phi_1| \\
&= \frac{4}{3} .
\end{aligned}$$

The example in Figure 6.1 shows that the bound $\frac{4}{3}$ is the best possible. SA might produce a list $L = (J_6, J_5, J_4, J_3, J_2, J_1)$. The corresponding SS equals 4 while OPS equals 3. ■

The following question is still open: What is the least upper-bound for ω_{SA}/ω_0 when there are 3 or more processors in the system?

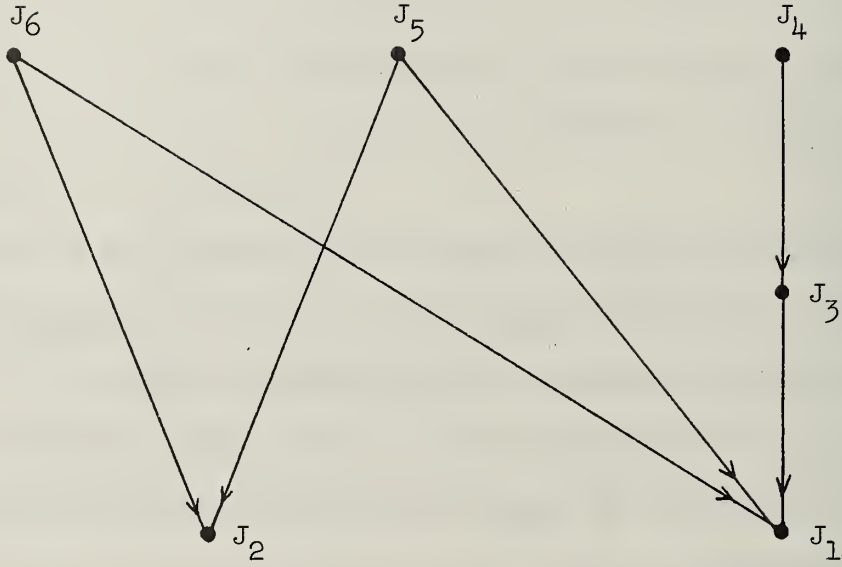


Figure 6.1 An example for which $\omega_{SA}/\omega_0 = \frac{4}{3}$.

CHAPTER 7

CONCLUSION

At this point there are certainly more questions than answers available. But the following questions seem to be particularly interesting.

1. What efficient algorithm A exists such that ω_A/ω_0 is bounded away from 2? It seems natural that if one is willing to use more complicated algorithms one can be guaranteed that ω_A/ω_0 will get closer to 1.

2. What efficient algorithms producing optimal schedules exist for a job set $(Q, \mu, \prec)$ with $\mu(J) = 1$ for all $J \in Q$ when there are 3 processors in the system.

LIST OF REFERENCES

- [1] Chen, N. F. and C. L. Liu, "On a Class of Scheduling Algorithms for Multiprocessor Computing Systems," 1974 Sagamore Computer Conference on Parallel Processing.
- [2] Chen, N. F. and C. L. Liu, "Bounds on the Critical Path Scheduling Algorithm for Multiprocessor Computing Systems," (to appear).
- [3] Coffman, Jr., E. G. and R. L. Graham, "Optimal Scheduling for Two Processor Systems," Acta Informatica 1,3 (1972), pp. 200-213.
- [4] Cook, S. A., "The Complexity of Theorem Proving Procedures," Proceedings of the 3rd ACM Symposium on Theory of Computing, 1970.
- [5] Fujii, M., T. Kasami, and K. Ninomiya, "Optimal Sequencing of Two Equivalent Processors," SIAM J. Appl. Math. 17,4 (July, 1969), pp. 784-789; Erratum 20,1 (January 1971), p. 141.
- [6] Garey, M. R. and R. L. Graham, "Bounds for Multiprocessing Scheduling with Resource Constraints," SIAM Journal on Computing, (to appear).
- [7] Garey, M. R., R. L. Graham, D. S. Johnson, and A. C. Yao, (to appear).
- [8] Graham, R. L., "Bounds on Multiprocessing Timing Anomalies," SIAM J. Appl. Math. 17,2 (March 1969), pp. 416-429.
- [9] Graham, R. L., "Bounds on Multiprocessing Anomalies and Related Packing Algorithms," Proceedings of the AFIPS Conference, 40 (1972), pp. 205-217.
- [10] Hu, T. C., "Parallel Sequencing and Assembly Line Problems," Oper. Res. 9,6 (November 1961), pp. 841-848.
- [11] Karp, R. M., "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher (Eds.), Plenum Press, New York, 1972, 85-104.
- [12] Lam, S. and R. Sethi, "Worst Case Analysis of Two Scheduling Algorithms," (to appear).

- [13] Liu, J. W. S. and C. L. Liu, "Bounds on Scheduling Algorithms for Heterogeneous Computer Systems," Proceedings of the IFIPS 1974 Congress, North-Holland Publishing Co., August 1974.
- [14] Ullman, J. D., "Polynomial Complete Scheduling Problems," 4th Symposium on Operating Systems Principles, Yorktown Heights, New York, (October 1973), pp. 96-101; to appear JCSS.
- [15] Yao, A. C., "Scheduling Unit-Time Tasks with Limited Resources," Proceedings of the Sagamore Computer Conference, 1974.

VITA

Nai-Fung Chen was born on May 22, 1945, in China.

He received a B.A. degree in Mathematics from the University of Hong Kong. In August, 1972, he received an A.M. degree in Mathematics from the University of Illinois at Urbana-Champaign.

While at the University of Illinois, he held a teaching assistantship for three years, a teaching fellowship for two years and a research assistantship for two years.

BIBLIOGRAPHIC DATA SHEET		1. Report No. UIUCDCS-R-75-724	2.	3. Recipient's Accession No.
4. Title and Subtitle AN ANALYSIS OF SCHEDULING ALGORITHMS IN MULTIPROCESSOR COMPUTING SYSTEMS				5. Report Date May, 1975
				6.
7. Author(s) Nai-Fung Chen				8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801				10. Project/Task/Work Unit No.
				11. Contract/Grant No. NSF GJ-41538
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.				13. Type of Report & Period Covered
				14.
15. Supplementary Notes				
16. Abstracts In this thesis the worst-case performance of three scheduling algorithms for unit-time jobs are studied. The least upper-bound on the ratio of the length of the critical path schedule to that of an optimal schedule is established for an n-processor system. A sufficient condition under which the critical path algorithm is optimal is given. A variation of the critical path algorithm is considered. It is shown to produce optimal schedules when there are only two processors in the system. Finally the successors algorithm is studied and the least upper-bound on its worst-case performance on a 2-processor system is obtained.				
17. Key Words and Document Analysis. 17a. Descriptors				
17b. Identifiers/Open-Ended Terms				
17c. COSATI Field/Group				
18. Availability Statement		19. Security Class (This Report) UNCLASSIFIED		21. No. of Pages 75
		20. Security Class (This Page) UNCLASSIFIED		22. Price

JUN 24 1975

SEP 5 1975

UNIVERSITY OF ILLINOIS-URBANA



3 0112 052127385